UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT ROVIRA I VIRGILI
UNIVERSITAT DE BARCELONA

MASTER OF ARTIFICIAL INTELLIGENCE THESIS

Incorporation of Models In Automatic Requirement Dependencies Detection

---

Author:
Ricard Borrull Baraut

Director:
Prof. Xavier Franch

Co-director:
Dr. Cristina Palomares

A thesis submitted in fulfillment of the requirements for the degree of Master of Artificial Intelligence

in the

Barcelona School of Informatics (FIB)
Universitat Politècnica de Catalunya

October 11, 2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT ROVIRA I VIRGILI
UNIVERSITAT DE BARCELONA

Abstract

Barcelona School of Informatics (FIB)
Universitat Politècnica de Catalunya

Master of Artificial Intelligence

Incorporation of Models In Automatic Requirement Dependencies Detection

by Ricard Borrull Baraut

Requirements Engineering (RE) is considered one of the most critical phases in software development. Usually it has to deal with Natural Language (NL) requirements documents of several hundred pages long, which are usually complex and full of dependencies between their requirements. The detection of these dependencies is crucial to ensure the reasoning and coherence of requirements documents. Nevertheless, the existent methodologies to detect those dependencies is usually undertaken with a manual and laborious process elaborated by experts stakeholders. Despite several works deal with the automatic detection of requirements dependencies, as far as we know, there are no approaches able to detect automatically different types of dependencies (e.g., refines, requires, incompatible, damages, etc.).

In this work, an approach to the automatic detection of dependencies in NL requirements is presented. The core of the approach is based on an ontology containing knowledge defining dependency relations between specific terminologies related to the domain of the input requirements. Additionally, in order to manage the information of these requirements, the approach uses several artificial intelligence techniques, such as Natural Language Processing (NLP) and Machine Learning (ML) algorithms. NLP techniques lie on syntactic and semantic text analysis to understand requirements clauses and extract their meanings. On the other hand, ML techniques are based on intelligent categorization, such as conceptual clustering, in order to classify requirements into the predefined ontology. The approach will be validated with data from an OpenReq partner (the project inside of which this thesis is being developed), Siemens AG Österreich, working in the railway systems domain.

Acknowledgements

This dissertation originated in cooperation with the Software and Service Engineering Group (GESSI), a research group of software engineering that belongs to the Universitat Politècnica de Catalunya (UPC). The work presented in this thesis has been conducted within the scope of the Horizon 2020 project OpenReq, which is supported by the European Union under the Grant Nr. 732463.

Special thanks to Xavier Franch Gutiérrez and Cristina Palomares Bonache for giving me the opportunity to cooperate on their project and guiding me through from beginning to end. In addition, I would also like to thank Siemens AG Österreich, an industrial company involved as a partner of the OpenReq project, for enabling the access to their data and knowledge used in this dissertation.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 OpenReq Project . . . 2
- 1.3 Motivation and Goals . . . 3
- 1.4 Methodology . . . 4
- 1.5 Organization . . . 5

2 State-of-the-art 6
- 2.1 Decomposition and Reduction of the Text . . . 6
- 2.2 Artificial Neural Network Model . . . 7
- 2.3 Syntactic and Semantic Analysis . . . 8
  - 2.3.1 Parse Graph . . . 8
- 2.4 Triplet Information Extraction . . . 9
- 2.5 Ontologies . . . 9
  - 2.5.1 Ontology Categorization by Word-Sense Disambiguation . . . 10
- 2.6 Topic Modeling . . . 10
- 2.7 Summarization . . . 12

3 Dependency Detection Approach 14
- 3.1 Project Overview . . . 14
  - 3.1.1 Data Quality Process . . . 15
  - 3.1.2 Data structure . . . 15
  - 3.1.3 Ontology Structure . . . 16
  - 3.1.4 Restrictions . . . 16
- 3.2 Methods and Algorithms . . . 17
  - 3.2.1 Preprocessing of the Data . . . 18
    - 3.2.1.1 Sentence Boundary Disambiguation . . . 18
    - 3.2.1.2 Noisy Text Cleaning . . . 19
  - 3.2.2 Syntactic Analysis . . . 20
    - 3.2.2.1 Tokenization . . . 20
    - 3.2.2.2 Part-of-Speech Tagging . . . 20
    - 3.2.2.3 Dependency Parser . . . 21
    - 3.2.2.4 Information Extraction . . . 22
    - 3.2.2.5 N-Grams Generation . . . 28
  - 3.2.3 Semantic Analysis . . . 29
    - 3.2.3.1 Lemmatization . . . 30
    - 3.2.3.2 Semantic Similarity . . . 30
  - 3.2.4 Ontology Categorization . . . 31
  - 3.2.5 Dependency Extraction . . . 32

4 Evaluation and Discussion **34**
4.1 Dataset . . . 34
4.1.1 Data Analysis . . . 34
4.1.2 Input Ontology . . . 35
4.2 Dependency Detection Results . . . 36
4.2.1 Validation of the Results by Experts . . . 38

5 Conclusions and Future Work **42**
5.1 Conclusions . . . 42
5.2 Future Work . . . 43

References **44**

A API Documentation **48**

B Glossary **51**
B.1 Glossary of General Acronyms . . . 51
B.2 Glossary of Part-of-Speech Tags . . . 52
B.3 Glossary of Dependencies from the Dependency Parser . . . 53

List of Figures

1.1 OpenReq high-level architecture diagram. 2

3.1 Ontology structure example. 16
3.2 Dependency detection approach diagram. 18
3.3 Dependency tree. 22
3.4 Pattern path examples. 23
3.5 Detected branches of the tree in the first rule. 25
3.6 Detected branches of the tree in the second rule. 26
3.7 Detected branches of the tree in the third rule. 27
3.8 Deepest path for term extraction example. 28
3.9 Different set of words situations to form n-grams. 29
3.10 Dependency link identification example. 32

4.1 Railway domain ontology graph. 36

List of Tables

2.1 Dependency type detected in researched studies. 12

3.1 Set of patterns detected in study case. 23
3.2 Dependency type of branches of first rule. 25
3.3 Dependency type of branches of the second rule. 26

4.1 Set of documents provided by Siemens 35
4.2 Detected dependencies within the provided RFPs. 37
4.3 Subset of requirement's terms and lemmas that matches with ontology concepts. 37
4.4 Precision measure of validated outcomes. 38
4.5 True positive dependency examples. 39
4.6 True positive dependency examples to be refined. 40
4.7 False positive dependency examples. 40

Chapter 1

Introduction

1.1 Context

Requirements engineering (RE) is a branch of system engineering which is recognized as an area of growing importance [1]. There exists an increasing effort devoted to research in this area, where many contributions attempt to solve different problems within it, such as RE modeling, tools to support RE process, reuse and adaptation of requirements, etc. One of them is *requirements traceability* (RT), which is defined as the "ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [2], hence throughout any phase of the system life cycle.

System traceability is a required component of the approval and certification process in most safety-critical systems, as in aerospace and railway domains [3]. On account of this, traceability information of requirements has to be monitored by recording, organizing and maintaining their knowledge, in order to allow their employment during the requirement's life, where it has also been observed that they mostly have dependencies between each other or between requirements and others development artifacts [4]. Consequently, if a requirement changes, traceability relations help to identify what other requirements may be affected by that change, so effective interdependencies detection and modeling is crucial for enabling the reasoning and coherence of requirements.

Thence, these dependencies play an important role in some industrial studies in a RE phase [5], [6]. Some properties can be useful to deal with these connections in requirements management and to improve the traceability accuracy. These properties are structural and semantic requirements features, which have been proposed to assist in identifying and classifying requirements relationships into dependency taxonomies, also known as typifications [7], [8].

As it is seen, requirements traceability is one of the most decisive process in a project [9], and as a consequence, traceability dependencies. However, despite its importance, dependency detection is usually a manual, laborious and elusive process, especially when the set of software requirements is large and specified in natural language (NL), which is the usual case [10]. The cost and effort needed to develop and maintain traceability links can be extremely high, and as far as it is known, there is even a lack of effective support tooling to deal with it [11], or as shown in Chapter 2, most of the studies to date are only focused on some types of traceability dependencies, such as inconsistencies.

In this work, we are going to focus on the automatic detection of traceability dependencies and its classification on a semantic level. Thus, the proposed approach aims to identify dependencies in NL requirements and cover several types of traceability relations between them (e.g., requirement A has a type of incompatibility with requirement B). To deal with it, Natural Language Processing (NLP) lies in the

core of the project to handle complex NL requirements, along with an ontology-based approach to extract and coordinate relations between them.

This master thesis has been carried out inside the OpenReq project and will be tested and validated with a dataset of one of the trials of the project team partners (Siemens AG Österreich). The data is based on a real railway request for proposal, where RE needs to be applied in order to ensure the correctness of the final service. However, as it is explained in section 3.1.2, the data is extracted from large amount of complex documents and needs a quality process to adapt the requirements into patterns or a more well-structured form, which is not a goal of this work, but of the OpenReq project.

1.2 OpenReq Project

OpenReq is an EU Horizon 2020 framework project [12] that aims to provide highly innovative methods, algorithms and tools for community-driven RE in large and distributed software-intensive projects. The project deals with the optimization process in RE areas as requirements detection, classification, monitoring and even decision making. Some of these areas requires artificial intelligence (AI) algorithms to give support to the identification, extraction and traceability process of requirements, which are written in large NL documents, and enable an optimal solution.

Therefore, OpenReq's architecture platform is composed by several modules capable of face all these goals and ensure an efficient result (figure 1.1).

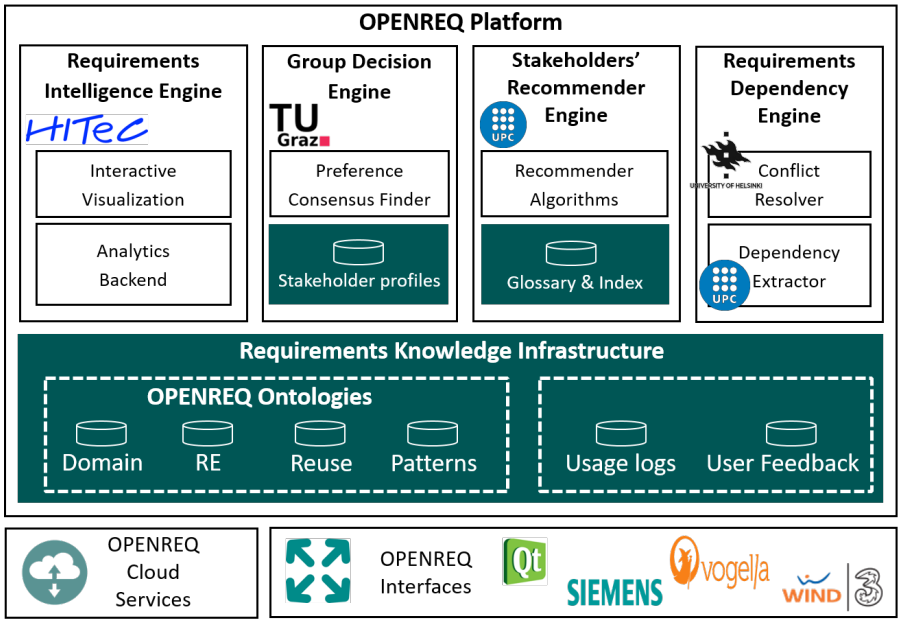


FIGURE 1.1: OpenReq high-level architecture diagram.

The architecture of the whole OpenReq project is composed of three main parts: the platform, the interfaces and the cloud services of OpenReq.

OpenReq Interfaces will provide open and unified interfaces for an easily integration of OpenReq into external tools in form of connectors. Otherwise, OpenReq Cloud Services collect and handle the massive amount of data, which will have virtually centralized data storage of high scalability that is easier to access, process, and manage.

As it is seen, OpenReq platform is formed by four principal components. The first one, which is called *Requirements Intelligence Engine*, is in charge of the automated identification of requirements from different knowledge sources.

The second module is the *Group Decision Engine*, which support decision making by providing solutions that fulfills all users preferences and identify conflicts between them, such as screening, negotiations or release planning of requirements.

Furthermore, the third component is the *Stakeholders' Recommender Engine*, which provides a set of standard queries to generate a recommendation, such as queries for new requirements, responsible stakeholders or even reusable requirements.

Finally, the fourth module is the *Requirements Dependency Engine*. This component is split in two main parts, the conflict resolver, which repairs inconsistent requirements, and the dependency extractor, which is the module developed in this work and is in charge of the automatic identification of dependencies with several NLP techniques and ontology knowledge.

During all this paper, the dependency extractor will be investigated, analyzed, developed and evaluated in order to support the detection and extraction of requirements dependencies in a semantic level.

1.3 Motivation and Goals

As it is mentioned in the introduction context (1.1), RE is considered one of the most critical activities in ICT projects. So, as much poorly RE is implemented, the greater will be the risk of a project failure [9]. Consequently, high quality requirements are an essential precondition for the success of an ICT project. Despite that, the research of this work shows that does not exist enough validated solutions to achieve OpenReq needs in relation to requirement traceability or to handle the whole RE problem. For that reason, the main motivation of this work deals with the development of an automatic dependency identification approach, which will ensure the detection of all taxonomies of requirement relations.

Bringing an artificial intelligence perspective to dependency detection field will provide an innovative approach to this research branch. It will help to solve complexity in the requirements understanding and know which is the type of the different requirements dependencies. In addition, it is a potential support to handle other RE modules that require requirements dependencies control, such as the reuse of requirements avoiding latent dependency conflicts, the rise of requirement information while mastering the complexity of requirements interdependencies, and the repair or concern of requirement modifications if it generates conflicts with some other linked requirements already established in the system.

Furthermore, the expected impacts with regard to this project in the RE field encourage and motivate its development in order to ensure an important innovation technology, which makes stakeholders more comfortable during the RE process as its efficiency is improved. Those impacts could be, among others, the reduction of the time to market, as this component will be able to give support into the automation of requirements traceability for large user communities that are engaged in the manual identification and selection of requirements to be implemented in their systems, or also the increase of productivity and quality of software projects by accelerating the development and maintenance of requirements and its linked requirement dependencies, and generating the automatic extraction of its knowledge to ensure quality requirements that are not conflicting between each other.

During the outline of this project, several questions arose to understand how to give a proper focus to that problem, such as *"Is it possible to handle all the dependencies relations?"* and *"How to identify and manage interdependency types in a given set of requirements?"* or even *"Which approach best supports stakeholders in requirements maintenance?"*. All these questions ensure a right focus into our research in order to achieve the goals of this work.

Thence, the ultimate objective would be: "*Develop a new approach to automatically detect dependencies between requirements*". Besides, in order to get closer to this goal a number of sub-objectives are defined:

- Determine dependencies between requirements on a semantic level.
- Take inspiration from other studies on monitoring requirements for requirements dependency management.
- Apply an intelligent approach based on artificial intelligence algorithms, as natural language processing.

On the assumption that the previous goals can be achieved at the end of this work, some ambition factors will appear as a consequence:

- An improvement in the RE process and significant efficiency gains.
- An achievement of a higher quality of requirements.
- A reduction of the time for inconsistency detection and consequently of investments in routine RE tasks.
- A reduction of the errors as a result of suboptimal RE processes, which is usually a manual task.
- A reduction of the risks to overlook critical dependencies or inconsistencies.

All of these reasons encourage this project to revolutionize the quality, coherence and efficiency of RE and ensure a real impact on future software projects.

1.4 Methodology

The methodology applied to elaborate this work lies in the correct implementation and operation of the OpenReq European Horizon 2020 project. We start from the basis that this project is a grant of initiation into research that has been proposed as a master thesis by the GESSI department in the UPC, which is a partner of the OpenReq project consortium.

Therefore, the project has several established restrictions by the OpenReq team, which we have to deal with. Moreover, they carry out some internal meetings to coordinate the work done in the different modules of the project and fix different formats, as for example the I/O of each project module. In order to handle with this requisites, weekly meetings are organized into the GESSI group in order to arrange all the work done during the life cycle of the project.

On the other hand, about the methodology applied in the decisions related to the method approach, a previous research of several articles that deals with the RE field had been collected and appraised during some months, in order to know which is the best solution to handle our problem and how can we deal with it. After that, some analysis tests of different methods had been done to evaluate their efficiency

and to know if they are the better solution to develop into our approach. At the end, a final decision of the approach was taken inside GESSI, where all the possible collected approaches were evaluated and discussed in order to know which of them can achieve the goals of the project.

Finally, after the approach was decided, the development process was initiated and several tests and evaluations were carried out during the development time in order to improve the results and ensure the best solution and accuracy of the dependency detection module.

1.5 Organization

This work has been organized into 5 separate chapters, including this introduction as Chapter 1. Chapter 2 provides the state of the art about previous studies done which apply different intelligent methods to our research field, such as NLP, artificial neural networks, ontology-based approaches, etc. Chapter 3 explains in detail the approach applied in this work, which includes data and project understanding and a deep analysis of the methods and algorithms used to solve the previous defined problems. Chapter 4 provides the evaluation and a discussion on the results extracted from several step points of our method, as tree graphs of parsing outcomes and triplet information extraction, and it terminates with a discussion on the final results. Chapter 5 concludes the work with a final statement of the results, the achieved objectives established in the beginning of the project and future work.

Additionally, there are two appendixes attached in this work. Appendix A provides the API Documentation of the dependency detection component, and appendix B provides the three glossaries of the acronyms used in this work.

Chapter 2

State-of-the-art

In this section, several RE literature on requirement dependency management filed are analyzed in order to know which methods and algorithms could be useful to apply in our approach, as well as to be aware of what applications exist to date for automatic dependency detection on a semantic level and which is their future work to be researched.

Thence, the research showed below will involve methods related to requirement simplification (2.1), such as a decomposition and reduction of the text, in order to get the most simple requirement to identify if their siblings have some dependency between them.

Another approach that will be involved in the research section is an artificial neural network model (2.2), which will compute their input data to extract requirement features and classify if there exists some dependency between pairs of input requirements.

Moreover, a research in NLP field is also established in this chapter, concretely a syntactic and semantic analysis (2.3), due to requirements are usually written in NL. Thus, it needs an exhaustive exploration of the text to avoid ambiguity and extract their terms meanings to allow a correct detection of requirement dependencies.

In addition, another approach of NLP called triplet information extraction (2.4) will be also studied in one of the following sections. Assuming that the the subject, verb and object of the clause are extracted, some dependencies can be detected by heuristic applications between those information.

Then, some articles use ontologies (2.5) to organize and store the extracted requirement information into a base knowledge, such as word-sense disambiguation, and apply exploration algorithms on ontologies, as semantic conflict analysis, to extract dependencies.

Ultimately, a model to extract topics (set of top words) from requirements (2.6) is studied in order to be able to obtain that terms that are relevant for requirements meaning, and detect dependencies with the support of some ontology approach.

Additionally, a brief analysis and the conclusions of the researched approaches are detailed at the end of these chapter.

2.1 Decomposition and Reduction of the Text

In order to deal with Inconsistency Detection (ID), articles [13], [14] propose a requirement improvement by a refining tree, which is generated by the decomposition and projection of a requirement as a finite AND/OR tree, in order to classify requirements as inconsistent or not.

Each node of the tree represents a basic requirement concept, where the OR node is generated by projection of the requirement in sub-requirements, and the AND node is generated by the decomposition of requirements in sub-patterns. So, the

requirements that have internal structure are called decomposable, and each of these subsystems has its own internal structure and it can be decomposed further until all the components have suitable problem patterns.

One of the main difficulties in the decomposable requirements is the projection nodes, which has to be transformed in an management of inconsistencies by the reduction of the OR node. Thus, they select one node among a group of OR nodes (with a tangent plane) to be used, and consequently the relations among the node in tangent plane tree become in an AND relation.

There exist different models to decompose the requirements:

- Top-Down functional decomposition (hierarchy structure): at each level, each function is decomposed into a number of functions at the next level.
- Use case decomposition: all actor interacts with the system to support a set of use cases and obtains an observable value result.
- Problem frame decomposition: first define a subproblem clauses and then use a projection of the full problem to recognize the subproblems.

Finally, in the last set of requirement trees, two concepts relations can be inconsistent if its entities result to a contradictory state between them.

2.2 Artificial Neural Network Model

An Artificial Neural Network (ANN) is a computing network system which is interconnected by nodes, also known as neurons, simulating a brain. Article [15] proposes a system based on a feed-forward neural network, where their main difference is based on its connections between neurons, which do not form a cycle.

Knowing that, they apply artificial neural network to learn contradiction relations from each pair of contradiction of the data. Note that these kind of relations are only one type of semantic of all that can exist.

To deal with that problem, the architecture of the ANN is comprised in six layers: lookup layer, convolutional layer, average-pooling layer, tanh layer, composition layer, and softmax layer. Thus, assuming that the input of the model are pairs of unaligned contradiction sentences, in the first four layers all the sentences and phrases are mapped into corresponding vectors in the same semantic space. Then, the sentence-level and phrase-level semantic relation representations are generated and concatenated along with three shallow features through the compositional layer. These features are the negation in the phrase, the difference of the word order and unaligned word number, which means the average number of unaligned words after removing overlapping words. Finally, the last layer works as a classifier to identify if the input sentences are contradictory.

Thence, in order to apply this method it is important to have a correct set of training data, such as, in this particular case, a data set formed by pairs of sentences with a contradictory relation between them. However, they don't prove that this method is going to work for requirement links which don't have an implicit relation, as refinement or conditional one.

2.3 Syntactic and Semantic Analysis

Dependency detection requires a highest level of text understanding. Due to that, text analysis is useful to manage requirements and detect its conflicts and dependencies. Syntactic and semantic analysis offers a good understanding of the text meaning which can be applied to requirements management.

Article [16] expose a NLP word analysis to detect inconsistencies between requirements' terms. Their approach is based in three steps: extract term corpus, syntactic aliasing and semantic analysis.

1. Extract term corpus:
   (a) Apply a text processing step to clan the requirement clause.
   (b) Identify and label each requirement with a unique identifier.
   (c) Individual tokens within each requirement sentence are annotated by applying Parts-of-Speech (POS) tagging techniques.
   (d) Term extraction by entity recognition and heuristics of standard language database.
   (e) Collect all the tokens as terms and rest individual words together with their tags
2. Syntax analysis:
   (a) Detect entity terms pairs as aliases, acronyms or short references names that refers to the same meaning entities.
   (b) Apply a semantically neutral phrase removal, where the phrases which do not play any active role in determining the reference to the entity it is referring to are eliminated.
   (c) Utilization of fuzzy variants, which measures the degree of dissimilarity between two words by the Levenstein distance metric, which is the smallest number of insertion, deletion, and substitution operations required to change one word into the other.
   (d) Apply abbreviation identification identifying short-forms and acronyms to avoid ambiguity.
3. Semantic analysis:
   (a) Generation of latent semantic model (LSA) to measure semantic proximity between terms.
   (b) Similarity computation between terms pair based on latent semantic, morphology, transitivity, and co-location similarity.
   (c) Generation of alias clusters to joint multiple entity aliases. WordNet is used to filter out false positives.

2.3.1 Parse Graph

Another NLP approach to analyze the syntax of a requirement is a parser process, which generates a syntax tree with POS-tags and relations between words. Article [17] uses trees as a dependency graph in order to extract dependency relations between a requirement and a conflict requirement hypothesis. Moreover, they use synonyms and named entities in order to improve the dependency graph. To deal

with this approach, the parse graph of the requirement and its conflict hypothesis text are generated and aligned, consisting of a mapping from each node in the requirement hypothesis to a unique node in the original requirement by a similarity scoring measure. Later, the pairs that can not be contradictory are filtered, in order to avoid false positive, knowing that the pairs that do not describe the same event can not be conflictive.

Consecutively, their contradiction features, which are described in detail in the mentioned article, are extracted, and finally, a logistic regression is applied in order to classify the resultant pairs as a real conflictive combination or not.

2.4 Triplet Information Extraction

Information extraction is used in some articles as a relevant part of the process to detect inconsistency and incompleteness of requirements within a large text document. That process is based on detecting sentence's relations between subject, verb and object, thus giving a basic and understanding context of the phrase. These relationships can be called SAO (Subject, Action, Object) or triplets.

Some studies deal with these process by similar generic algorithms [18], [19] with the objective of extracting these relations.

In order to extract these relations, a parser model is needed, which generates a syntax tree tagged with several POS-tags. Once the POS-tagging and parser process are applied, and assuming that the analyzed sentence is basic and well written, in the parse tree can be found two main branches, one for a noun phrase, and another one for a verbal phrase. The first branch contains the subject of the sentence, which is formed by the first noun of the branch and its siblings attributes followed by some prepositional phrase (if it has it). On the other hand, the verbal phrase will have the action of the sentence, followed by the object that affects the verb. The action is extracted by the first verb of the branch and its siblings adverbs (in case it have them), followed by some uncle verb in the tree. Otherwise, the object is the first noun that follows the predicate (verb) within the verbal phrase, its siblings attributes and if it is the case a prepositional phrase.

Once the data is extracted in a SAO relation form, it can be treated with different approaches. Article [20] proposes a conflict detection management by a syntactic approach, which is rule based. They assume that verb and object are used to detect and analyze activity and resource conflicts, so their method identifies candidate requirements conflicts by the following predefined conditions:

- Different Verb $\cap$ Same Object $\rightarrow$ Activity conflict,
- Same Verb $\cap$ Different Object $\rightarrow$ Activity conflict,
- Same Resource $\rightarrow$ Resource conflict.

With a different focus, article [21] proposes a SAO-based content analysis where suitable dictionaries of verbs and terms, which are related to a topic of interest, are used in order to assigning a score to each SAO-triplet to determine how much this requirement deals with the previously mentioned topic.

2.5 Ontologies

Organize and order the information is an important fact of requirements management. Several studies work with support of ontologies and graphs as a knowledge

base in order to save and create connections between requirements or its concepts, which can mark dependency relations.

Articles [22], [23] work with generic ontologies which can be used to indicate software characteristics and organizational elements, as well as, for example, store the information of different types of requirements formats and the information that each of them contain, as the action, agent, time, etc. To deal with it, article [23] uses a semantic graph represented as an OWL ontology and works with suitable specific glossaries which contains terms that allows them to create instances and relationships between requirements.

On the other hand, article [24] works with project-specific terms ontology classes, which allows them to manage the project requirements in a deep way. To deal with it, the article develops an instance fetcher which takes input data, analyses their contents and assigns their content into the categories of the ontology. This approach is based in a semantic point of view categorization, which applies NLP algorithms and some AI-heuristics, and finally analyze different types of inconsistency dependency, such as antonym, negation, numeric, etc. The following steps show the process of their approach:

1. Build the ontology by defining the requirement classes from specific term of the glossary.
2. Preprocessing if the information by a stop-words removal after providing the input data.
3. Steaming the outcome words to get a generic term.
4. Get the synonyms and hyponyms of the generated words to avoid false negatives.
5. Apply heuristic-based assignment of each requirement to the defined ontology categories, depending on the number of matches between the category's information and the extracted data, as for example the synonyms, steams, etc.
6. Apply semantic conflict analysis, which assumes that the requirements are written in specified grammar patterns, as EBNF requirements template, so it can be analyzed the parts of the pattern to extract inconsistencies between requirement relations.

2.5.1 Ontology Categorization by Word-Sense Disambiguation

The categorization process of the extracted text features into an ontology is not an easy procedure. In order to deal with that problem, the research of this problem is focused in Word-Sense Disambiguation (WSD), following the idea that each category is a label and we want to cluster requirements' context to them.

Some studies [25][26][27][28] explain different approaches of WSD in order to identify the correct sense of the nouns. All of these studies follow the same basic idea, the utilization of the WordNet ontology/database to extract lexical meanings of every word to disambiguate its sense.

2.6 Topic Modeling

Topic modeling is a type of unsupervised statistical model that attempts to describe a set of observations as a blend of categories. The most known method for this approach is Latent Dirichlet Allocation (LDA) [29], which is a generative probabilistic

model used to discover a specified number of topics that occurs in a collection of documents within a corpus, where text features are the occurrence count of each word. Documents are represented as random mixtures over latent topics, where each one is mined as a probability distribution over words. Its basic process assumes that the generative model choose word-by-word a topic mixture for each document.

Therefore, after modeling the document lengths as Poisson distribution (step 1), and generate a topic probability from a Dirichlet distribution (step 2), the algorithm will assign every word to a temporary topic by a multinomial distribution determined by the previously generated θ (Step 3a). Straightaway, topic assignments will be checked and updated, looping through each word in every document, based on the prevalence of that word across topics, and the prevalence of the topics in the document (Step 3b).

1. Choose $N \sim Poisson(\xi)$
2. Choose $\theta \sim Dirichlet(\alpha)$
3. For each of the N words in a document (w_n):
   (a) Choose a topic $z_n \sim Multinomial(\theta)$.
   (b) Choose a word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

Where:

- N is the number of words that document contains,
- θ is the probability that a randomly selected a word belongs to a topic $i \in \{1...k\}$,
- α is a prior estimate on topic probability, which means the average frequency of each topic occurs within a document in a corpus,
- β is a collection of k topics where each topic is given a probability distribution over the vocabulary used in a document corpus.

Note that the algorithm is used for long documents and to extract contexts in order to be able to know about them. However, for the problem of this work, requirements contain much information which is lost in that kind of general process, so it is needed to extract the topics from sentences and not from the entire document. Articles [30] and [31] shown an extension of LDA, which incorporate a new frame for a sentence level which would avoid that limitation.

Sentence LDA extension assumes the following overcome process for each document w in a corpus D:

1. Choose $S_d \sim Poisson(\xi)$
2. Choose $\theta \sim Dirichlet(\alpha)$
3. For each sentence $s \in [1, S_d]$ in a document (w_n):
   (a) Choose number of words $W_s \sim Poisson(\xi_d)$
   (b) Choose topic $z_{d,s} \sim Categorical(\theta)$
   (c) For each word $W_{d,s}$ in a sentence (s):

i. Choose a word $w \sim Multinomial(\phi_{z_{d,s}})$

Knowing that, in LDA sentence extension, the joint multinomial distribution is factored as:

$$p(w, z|\alpha, \beta) = p(w|z, \beta)p(z|\alpha)$$

Finally, when requirement topics are extracted, some comparative approaches, as word-sense disambiguation and ontology-based methods, which are explained in previous section, can be applied in order to extract its dependencies.

2.7 Summarization

Encouragingly, there appears to be some ongoing research into discovering solutions around the problem of text dependencies identification, such as antonyms, lexical contradictions, numeric incoherence, etc. Notwithstanding, it does not seem to be a large research established in automatic requirement dependencies identification within the RE field, much less in a complete solution of the whole problem already discussed in section 1, as they can not identify different types of dependency, which are detailed in section 3.1, that our approach aims to detect (table 2.1).

	Dependency Types							
	Contributes	Damages	Refines	Requires	Incompatible	Similar	Duplicates	Replaces
Decomposition / Reduction [13]					X			
ANN [15]					X			
Text Analysis [16]					X			
Parser Graph [17]					X			
Triplet IE (rules) [20]					X			
Triplet IE (comparison) [21]					X			
Semantic Text OWL [23]					X			
Project Term OWL [24]					X			
Proposed Approach	X	X	X	X	X	X	X	X

TABLE 2.1: Dependency type detected in researched studies.

Although the studies that have been found only detect incompatible type of dependencies, also known as inconsistency or conflicting dependencies, some of those approaches, as in the case of ontology-based ones, can be an inspiration to achieve our dependency detection objectives. In addition, other methods of NLP, such as a syntactic and semantic analysis, a parser tree or triplet information extraction, can be used for the management of the requirements written in NL and extract information that can be useful to detect dependencies on a semantic level.

Otherwise, some studied approaches, such as ANN and parser graph, require an specific format of the input data, such as a labeled set of requirement pairs, to learn how a dependency between two requirements is, in order to detect them with their trained model. However, this particular kind of labeled data is not owned for the OpenReq project, but a set of documents with a huge number of NL requirements.

Thus, supervised methods, such as ANN, are not useful to deal with the goals of our work, as our input data requires an unsupervised approach to achieve the expected outcomes.

To sum up, no study to date, as far as we know, reach to detect different types of dependency between requirements, or even more than one that is not an incompatible dependency. Thence, this work can be a huge challenge and a great opportunity for the research in the field of RE and dependency detection.

Chapter 3

Dependency Detection Approach

The proposed approach to deal with the automatic requirements dependency detection on a semantic level is presented in detail in this chapter. Additionally, a brief overview of the dependency types, the data context, and the restrictions of the project are analyzed to understand the challenge we have to cope with.

3.1 Project Overview

The major factor on which this work is based is the potential ability to detect different types of the dependencies. Thence, first of all it is necessary to know which are the semantics of the dependencies that are used within the OpenReq project and how these types of dependencies can be useful to refine a requirements document.

Therefore, a number of semantics are defined in order to understand the need of these dependency types. Note that the following types are the ones used in this work and in the OpenReq project to date, but they can be easily extended and fitted to the proposed approach with the support of the input ontology.

- Contributes: a requirement has a positive influence to another requirement.
- Damages: a requirement has a negative influence to another requirement.
- Refines: a requirement or target object is defined in more detail by a more specific requirement.
- Requires: a requirement is demanded by another one but not vice versa.
- Incompatible: requirements either conflicting with each other or with some policy or business rule. They cannot exist at the same time.
- Similar: a requirement overlaps between one or more requirements.
- Duplicates: a requirement is repeated one or more times by other requirements.
- Replaces: a replacement of a requirement with another.

All these dependency types can be found in a set of requirements due to numerous factors that can occur throughout the creation of request for proposals during the RE process. The major factor is caused by the defective coordination between all the stakeholders that aim to write their own requirements without taking into account the proposals of the other agents, generating damages, incompatibility, similar requirements, etc.

Additionally, these datasets of requirements are usually complex, hard to understand and sometimes even faulty, due to the fact that these NL requirements are

manually written by multiple stakeholders, which is a laborious challenge, resulting in a bad written of documents and requirements.

3.1.1 Data Quality Process

Assuming the complexity of the data and its unsuitable writing to manage its information, a quality process is required in order to organize the data and refine its requirements.

The OpenReq team is working on the development of several API functionalities to optimize the quality of the requirements for improving the accuracy of subsequent steps, such as the traceability of the requirements and dependency detection, by avoiding ambiguities and generating a controlled text environment. Part of this environment is created by the adaptation of the requirements into a template, which is a blueprint for the semantic structure of individual requirements [32].

Thence, those functionalities lies on different NLP techniques to give suggestions on how to improve the quality of NL requirements. Thus, *lexical checker* is the functionality that examine individual requirement's terms and word combinations against a glossary to check for ambiguities. These ambiguities include dangerous plurals, imprecise words, weak words, pronoun misuse, etc.

RegExps checker test requirement's word combinations and phrases using glossaries of regular expressions, which are previously detailed by experts, in order to detect ambiguities. These ambiguities could be unclear inclusion, ambiguous plurals, and unclear associativity, which are included in the glossaries.

Then, *PoS-RegExps checker* analyze requirement's word combinations and phrases using regular expressions glossaries designed to support NLP that include the part-of-speech (PoS) tags of their terms.

Ultimately, *templates conformance checker* proof that the requirement text follows a requirement template. Some examples of the templates used for checking the conformance are:

- <*system name*> shall/should/will <*process*> <*object*>
- <*system name*> shall/should/will provide <*whom*> with the ability to <*process*> <*object*>
- <*system name*> shall/should/will be able to <*process*> <*object*>

It is important to recall that apart from these templates, there are another ones that can be used in the quality template conformance process, which have been identified reviewing different literature sources on well-practice writing guidelines for requirements [32], [33], as the examples listed above.

3.1.2 Data structure

Another important aspect of the data is the input and output (I/O) format to be able to be used in this work. Thus, as this component is a module of a bigger project, the consortium of the OpenReq project decided that the I/O data has to be in JSON format, following an inclusive syntax that can deal with all the components of the whole project. Therefore, the following tags within the JSON format are required for our dependency detection component:

- **"projects"**: List of projects (JSON object) where each one implies the id of the project and the id's of the specified requirements within it.

- **"requirements"**: List of requirements (JSON object) where each one demands an id and a textual clause.
- **"dependencies"**: List of dependencies (JSON object) where each one requires the id where the dependency comes from, the id of the requirement to which the dependency refers, the taxonomy (semantic) of the dependency and its status (e.g., proposed, accepted, etc.).

Nevertheless, the structure of the requirements data would mean nothing without a correct structure of the input ontology, which should be provided by expert stakeholders.

3.1.3 Ontology Structure

As the component of this work follows an ontology-based approach, it is really important to understand how the ontology knowledge is structured before analyze the methodology of the approach.

Thence, this ontology is formed by several classes, where each one encapsulates a technical concept regarded to the data, which in our case is related to the railway domain, such as "ETCS", "Railway Switch" or "Train Protection System", among others. All these concepts are going to be involved in one or more dependencies between another concept by an ontology object property, which has the meaning of the dependency type between both concepts (figure 3.1).

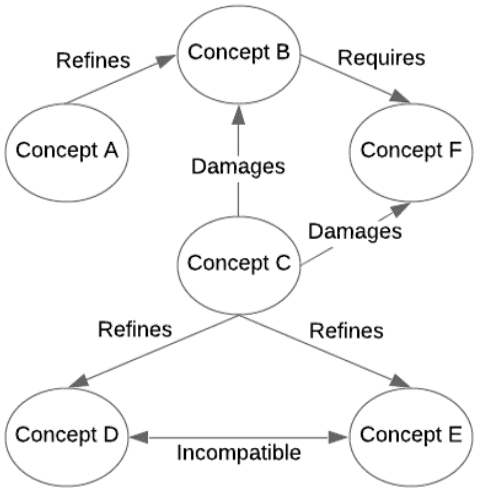


FIGURE 3.1: Ontology structure example.

3.1.4 Restrictions

As this work is a component part of the bigger project of OpenReq, some constrains have to be taken into account at the time of developing the approach and select the needed external frameworks.

In this particular case, the external frameworks used in our component have to be compatible with the license of EPL-v1.0[1], which is the one used in the OpenReq project.

[1]Eclipse Public License - v1.0, https://www.eclipse.org/legal/epl-v10.html

Furthermore, this work also has to deal with a restriction of licenses and permissions. Concretely, the GPLv3[2] must not be used in the component dependencies or derivatives.

3.2 Methods and Algorithms

In order to achieve the objective of being able to detect dependencies between requirements and know of which type are them, the approach shown in figure 3.2 has been followed. It is based in five main parts which include NLP methods to treat the text data and extract its information, and ML algorithms to categorize the extracted requirements into the ontology in order to identify its dependency types and return the final outcome.

This approach was decided after an extensive study period of ideas of proposals. All mentioned articles in chapter 2 were analyzed, and some ideas of each methodology were extracted and used as inspiration to propose a final approach that can deal with the goals of this work.

Thence, some methods as ontology-based ones, which define ontology classes as project technical terms and connect them with each other to detect different types of the incompatible dependency, were taken as inspiration to store knowledge in the form of domain terms, and link each class in a dependency type level. Then, to be able to store each requirement in a concept term class, it was required some NLP approach to manage and extract the concepts that can be matched with the ontology classes, such as text analysis methods, which different methods within the studied articles was also taken as inspiration, and they evolved to conclude in our final proposed approach. In the following sections of this chapter, all the methods and algorithms that deal with the defined problem are exposed.

Figure 3.2 shows the sequence of the steps that are proposed in this work. Reading the input data is the first step that is needed to start the approach, and where two types of data are required. The first one is the set of requirements that are going to be analyzed during the identification of the dependencies. The second type of input data is the ontology knowledge, where each class that forms it is named with an explicit term of a specific domain in which the data is based, as it was previously mentioned in the ontology structure (section 3.1.3). That ontology should be created by expert stakeholders that know which are the technical terms of the data that can have implicit and explicit dependencies. In our case, the input ontology is created by experts from the OpenReq team.

After that, a preprocessing step (section 3.2.1) to treat and clean the data is needed in order to analyze and manage the data without misunderstandings and parsing errors. This step has a critical importance to improve the results as it cleans the data in order to be well-read in the syntax analysis step. In addition, this process is going to be more sophisticated with the support of the quality process of the OpenReq project (section 3.1.1), which is in development process and it can not be used yet.

Syntax analysis (section 3.2.2) is the step where the greater number of NLP methods are applied in order to extract terms that are potential candidates to be matched with the knowledge ontology. Two main methods are established in this step, the dependency parser, where relations between words are detected, and the information extraction (IE), where the potential candidates are identified based on the parser relations.

[2]GNU General Public License - v3, https://www.gnu.org/licenses/gpl-3.0.html

Then, semantic analysis (section 3.2.3) treats the ambiguity of terms by applying different algorithms as lemmatization and semantic similarity with WordNet, allowing to match the extracted terms of the requirements with the terms of the ontology in the case they are written in a derivative form or are a synonym of a term in the ontology.

Finally, a categorization of the extracted terms into the ontology classes (section 3.2.4) are done with a conceptual clustering method, knowing the extracted features of the previous steps. Immediately, the dependencies between requirements are extracted by the relations of the ontology (section 3.2.5), allowing to know the different types of dependency and return the outcome of the component.

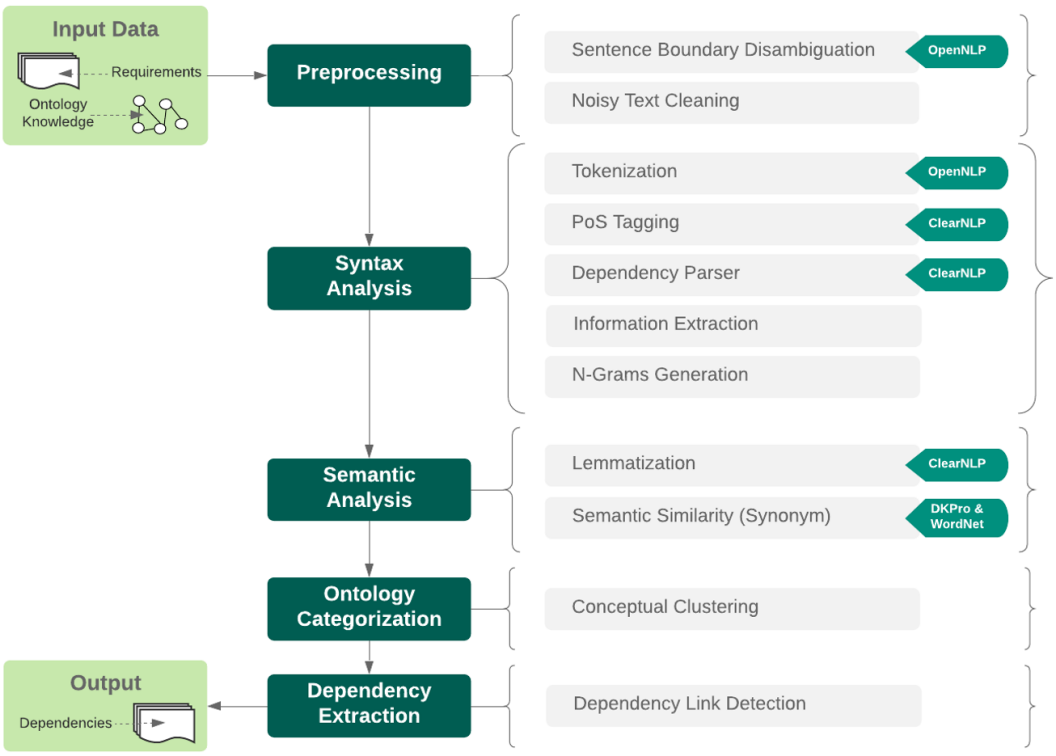


FIGURE 3.2: Dependency detection approach diagram.

In the following sections, each step of the approach is explained in detail in order to know how each algorithm is, why they are used and how they can achieve the goals of the project.

3.2.1 Preprocessing of the Data

Once the set of requirements are read, it is important to apply a preprocessing process to clean the data because, as it is previously mentioned, the data has numerous deficiencies that need to be treated in order to avoid misunderstandings in the syntax and semantic analysis. Thence, several methods and rules are applied in order to be closer to have requirements that are as much understandable as possible.

3.2.1.1 Sentence Boundary Disambiguation

First, remembering that the requirements can be formed by a set of sub-requirements, the first method that is applied is a sentence detection to split each phrase of the

whole input requirement, in order to optimize the accuracy of the following analysis steps. This process is also known as sentence boundary disambiguation (SBC) and lies in the process of deciding which is the beginning and the end of a sentence, knowing that a punctuation character may mark the end of a sentence or not.

Thence, in order to realize this sentence detection, the support of the framework Apache OpenNLP[3] is used, which is a library based on machine learning algorithms to process NL, such as tokenization, sentence segmentation, part-of-speech tagging, parsing, etc. This framework has several trained models for each algorithm, and in this case, the predefined model is trained to detect sentences in a given raw text, and the rule to define a sentence used by OpenNLP is described as "*the longest white space trimmed character sequence between two punctuation marks, except the first and last sentence of the text, which are the first and last non whitespace character, respectively*" [34].

3.2.1.2 Noisy Text Cleaning

Secondly, after the SBD algorithm, a method to clean the noisy text of each segmented sentence is applied. Thence, in order to accomplish a correct cleaning of the text, an ordered sequence of rules are applied:

1. Removal of tabulation spaces at the beginning of the phrase.
2. Removal of acronyms between parenthesis that are outside of the end of the sentence (e.g., the pattern: "*«sentence» «end punctuation character» («Acronym»)*" that can be found in the example requirement: "*[...] the balises. (LC)*") to prevent parser endpoint faults.
3. Removal of non word character list pointers (e.g., "*", "-", "•").
4. Removal of word character list pointers (e.g., "a)", "B.").
5. Removal of numeric list pointers (e.g., "1)", "1.2.").
6. Removal of roman numerals list pointers (e.g., "i", "vi", "ix").
7. Removal of id acronyms that may appear at the beginning of the requirement (e.g., "RBC 1", "NOTE 2").
8. Replacement of scape sequences characters by its real NL character (e.g., "\n", "\t", "\/").
9. Addition of a white space before a possessive expression (e.g., *«word»'*s) to prevent PoS tagger faults.
10. Addition of white spaces between parenthesis to prevent PoS tagger faults.
11. Addition of white spaces between quotation marks to prevent PoS tagger faults.
12. Addition of white spaces between point characters in the middle of a sentence if the SBD do not split the requirement phrase correctly to prevent parser faults.
13. Replacement of double white spaces to one white space.
14. Addition of an endpoint if there is none punctuation character at the end of the sentence.

[3] Apache OpenNLP, https://opennlp.apache.org

Note that all the removed or replaced issues are not important at the time of analyze and extract the top words of requirements, so it clarifies and refines the sentence.

Furthermore, all these rules are critical steps to be done before applying the part-of-speech (PoS) tagging and the parser, as it avoids textual faults that may occur with the utilization of NL requirements, as wrong tokenization caused by the wrong split of phrases between parenthesis or quotation marks without white spaces, which evolves in a bad PoS tags.

3.2.2 Syntactic Analysis

At this point of the approach, assuming that each requirement is segmented in sub-requirements or sentences, and each one of them is refined by applying the noisy test cleaning method, where unnecessary parts of the sentence are cleared, the syntactic analysis can be applied to extract those words that are potential candidates to be matched with concepts of the ontology.

The application of a syntactic analysis approach has been required in order to analyze the grammar of each requirement, obtain information about the relation of each word within the whole sentence set, be aware of the important parts of the requirements and be able to extract requirement keywords or concepts and summarize its meaning in a set of top words.

To deal with this task, two main methods are developed in the approach, the dependency parser and the IE, which at the same time they require a set of methods to be applied.

3.2.2.1 Tokenization

First of all a treatment of the sentence is needed to get the required form of the text to apply the subsequent methods. Thus, a tokenization of the sentence is used in order to split the input sentence into single words.

This step is done with the support of the OpenNLP framework, which has a trained model to tokenize raw text sentence based on a set of delimiters, as white spaces, character classes and boundaries based on a probability model which uses maximum entropy to take its decisions. In our work, a probability model is used to the tokenization step.

Sentence:	"The parameters for OBU mut be given by RBC."
Tokens:	"The", "parmeters", "for", "OBU", "must", "be", "given", "by", "RBC", "."

3.2.2.2 Part-of-Speech Tagging

Therefore, once the tokenizer has been used, ensuring that the produced tokens are the expected ones by the previous preprocessing step of clearing the data, a PoS tagger, which works with a set of tokens, can be immediately applied.

A PoS tagger marks tokens with their corresponding word type based on the token itself, where each token might have multiple choices of PoS tags depending on its context, thus the complexity of this process lies on the hard decision of which tag

suits better each token. Note that the acronyms of PoS tags shown in the following example are detailed in appendix B.2.

The	parameters	for	OBU	must	be	given	by	RBC	.
DT	*NNS*	*IN*	*NP*	*MD*	*VB*	*VBN*	*IN*	*NNP*	.

To deal with this process, the software tool DKPro[4] is used. This tool allows join frameworks and resources, such as OpenNLP, ClearNLP and WordNet, among others, to reuse NLP components and permit their functionalities combination to achieve an efficient NL analysis. In our case, DKPro gives support to apply the ClearNLP[5] framework, which is an application of supervised machine learning to NL data processing, such as lemmatization, PoS tagging, parsing, etc. Therefore, the ClearNLP PoS tagger is in this step of the component, which uses the generalized model from a dynamic model selection and utilizes ambiguity classes trained on a large corpus to obtain its outcome [35].

3.2.2.3 Dependency Parser

The dependency parsing is one of the most important methods of this step, together with IE. It analyzes the grammatical structure of a sentence, establishing relationships between the parent words and the child words. For that reason, it is important to apply a sophisticated parser that can deal with the NL requirements.

Thence, once each word has been tagged, the dependency parser can be run to get more information of the words by extracting their relations. In our case, the same support tool used in the PoS tagging, which is named DKPro, is used to apply the ClearNLP dependency parser, which uses a transition-based, non-projective parsing algorithm showing a linear-time speed for both projective and non-projective parsing. This means that the outcome of the parser will be a syntactic ordered set of tokens with discontinuous syntactic relations of their dependencies.

The approach of a general transition-based dependency parsing lies on a linear-time scan over the words of a phrase, where at every step it pushes the next sequential word to be processed over a stack, whose base has to be the root of the sentence, maintaining the rest of the ordered words in a buffer. Thus, assuming that the initial state is to have all the words in order on the buffer, an empty stack with a dummy root and an empty set of dependency arcs, the transitions of this method to generate the arrows are [36], [37]:

- Transitions Left-arc: add a dependency arrow to the set of arcs from the first word of the buffer to the top word of the stack. Pop the stack if the word is not the root of the sentence.
- Transitions Right-arc: add a dependency arrow to the set of arcs from the top word of the stack and the first word of the buffer. Pop the stack if the word is not the root of the sentence and replace the used head word of the buffer with the removed word from the stack to reprocess the word.
- Transition Shift: Removes the head node of the buffer and push it to the top of the stack.

[4]DKPRO, https://dkpro.github.io/
[5]CoreNLP, https://emorynlp.github.io/nlp4j

Therefore, these transitions allow the parser to generate dependencies between words, where each dependency type is proposed depending on the PoS tags of the analyzed pair of words.

Knowing how the dependency parser works, it is important to recall that the outcome of the ClearNLP parser is a sequence of non-linked collection of dependencies, where each node of the collection represents a token with its dependency related to the token of its parent. Thus, in order to work with parser outcomes and apply a correct method of IE, a generation of a dependency tree is implemented, where each node has linked attributes of its direct parent and its direct children.

Figure 3.3 shows a dependency tree extracted from the parser results of a simple example phrase. Each node of the tree represents a token with its PoS tag, and each arrow shows the dependency link between the node itself with its parent, who is the one that receives the dependency. In our case, the non-linked collection of dependencies that the parser returns has evolved in a set of linked nodes by its dependencies generating that kind of tree. Thence, knowing how is the structure of the dependency tree, an analysis of its words and dependencies can be done in order to extract information related to our approach. Note that the abbreviations of the dependencies (figure 3.3) of the dependency parser are detailed in appendix B.3.

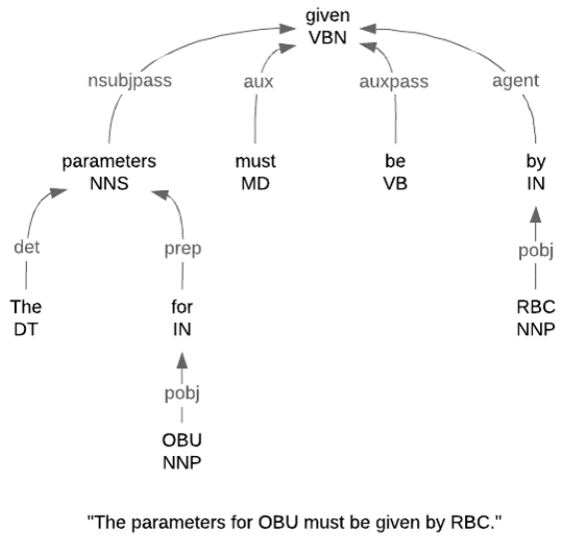


FIGURE 3.3: Dependency tree.

3.2.2.4 Information Extraction

Once the form of the data is parsed to an optimal form to be analyzed (i.e., a dependency tree form), the needed information to categorize each requirement into the ontology classes can be extracted. This information is the keywords of requirements that are considered potential candidates to have a match with the ontology.

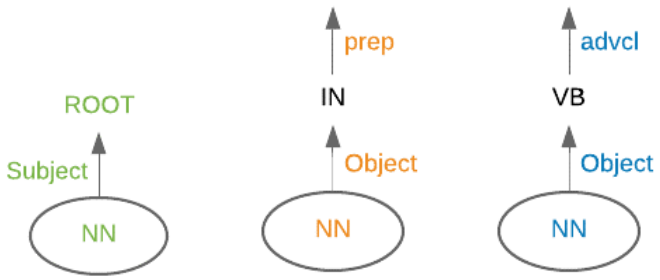
In order to extract these keywords, a great number of requirements were manually analyzed by Siemens' experts to extract the keywords of each requirement, allowing to know the potential keywords' candidates. Then, once those keywords were known, the dependency tree of all the sentences of these requirements were manually analyzed in order to find a pattern of the position of these keywords within them. This position is defined by the dependency path (i.e., path of links of the dependency tree) and the PoS tag of each keyword within the dependency tree.

Thence, after an exhaustive study of the dependency trees, these patterns were detected. Despite several of those patterns were identified numerous times in different requirements, some other ones only appear in particular cases, resulting in irrelevant patterns that can be discarded for the correctness identification of the keywords (using those not so frequent patterns would have introduced more irrelevant keywords when applied in other requirements than relevant ones). Thus, only the most probable patterns to success in the search of potential keywords' candidates were taken into account in the implementation of the information extraction rules.

Pattern	**Path Definition**
1	Subject(NN) - ROOT
2	Subject(NN) - Adverbial clause modifier
3	Object(NN) - Preposition
4	Object(NN) - Agent
5	Object(NN) - Complement
6	Object(NN) - Adverbial clause modifier
7	Object(NN) - Preposition - Conjunct
8	Object(VB) - Preposition
9	Conjunct(NN) - Preposition
10	Noun compound modifier(NN) - Subject
11	Noun compound modifier(NN) - Object
12	Noun compound modifier(NN) - Adjectival modifier
13	Appositional modifier(NN) - Object
14	Appositional modifier(NN) - Subject
15	ROOT(NN)

TABLE 3.1: Set of patterns detected in study case.

Table 3.1 shows the set of patterns (extracted from studying the requirements) used for the location of every keyword within its dependency tree. These patterns are based on the branch tree path that the keyword can reach. For example, pattern 1 (figure 3.4a) shows that a keyword can be located as the direct subject noun of the sentence that reaches from the root, or, as it is seen in patterns 3 and 4 (figures 3.4b and 3.4c) , objects nouns can pass through a preposition or an agent without taking these non-direct dependency nodes as word candidates. It is important to recall that the PoS tag of the candidate keyword has to be taken into account, while the other terms that are in the path from the keyword to the root are not relevant to the pattern.



(A) Pattern 1. (B) Pattern 3. (C) Pattern 4.

FIGURE 3.4: Pattern path examples.

Additionally, as these patterns are extracted from a study case, we have to assume that there can be more general dependency paths that can deal with our requirements' keywords.

In order to ensure a correct extraction of all these patterns, they have been adapted into three rules. These rules lie on a branch analysis, where three kind of level decisions can be applied depending on the dependency type of each one. These three decisions, or rules, are: analyze the branch and omit the node (from now on, first level rule), analyze the branch and get the node as a potential candidate (from now on, second level rule), or directly prune the branch (from now on, third level rule).

Thence, the three rules to be applied in each node of the tree in order to identify potential candidates are sequentially defined in three levels:

1. *First level rule.* If a descendant branch of the node contains a non-direct but relevant dependency type to a term candidate, the child node is analyzed but the child term is not marked as a candidate.

2. *Second level rule.* If the previous rule (i.e., first level rule) is not applied, verify whether a descendant branch of the node contains a direct relevant dependency type to a term candidate. If it is true, the child node is analyzed and the child term is marked as a candidate.

3. *Third level rule.* If none of the previous rules is applied, it means that the descendant branch contains an irrelevant dependency type, so the brunch is pruned and not analyzed.

In the following sections, these rules are explained in detail in order to understand the correct functionality of the IE step.

First Level Rule

The first rule is based on those dependency types of the descendant branches of one node in which the direct term of the node is omitted as it is not considered as a potential candidate, but its branch is analyzed to look for more descendants by applying all the rules. These dependency types can be related to the non-direct but relevant paths from where the candidate reach within the pattern, such as prepositions, agents, complements, etc.

In order to be closer to achieve that rule, dependency branch types taken into account in this work are listed in table 3.2.

Note that the list of table 3.2 can be reduced or extended easily to avoid or take into account other branches.

Figure 3.5 shows the branches of the dependency tree, which are marked with orange color, which would be taken into account in the first rule for a specific requirement. In that example, the root node is analyzed within the dependency 1 of the first level rule table (table 3.2), and it can be detected as a non direct dependency in the pattern 1 of the patterns table (table 3.1). The "prep" branch its detected with the dependency 3 of the first level rule table, and it can be seen as a non direct dependency in the pattern 3 of the patterns table. Finally, the "agent" branch its detected with the dependency 4 of the first level rule table, and it can be detected as a non direct dependency in the pattern 4 of the patterns table.

Dependency	Branch Type Definition
1	ROOT
2	Object predicate
3	Preposition
4	Agent
5	Adjectival modifier
6	Adverbial clause modifier
7	Clausal complement with external subject
8	Clausal complement with internal subject
9	Adjectival complement
10	Prepositional complement
11	Possessive
12	Participial modifier
13	Passive clausal subject
14	Conjunct

TABLE 3.2: Dependency type of branches of first rule.

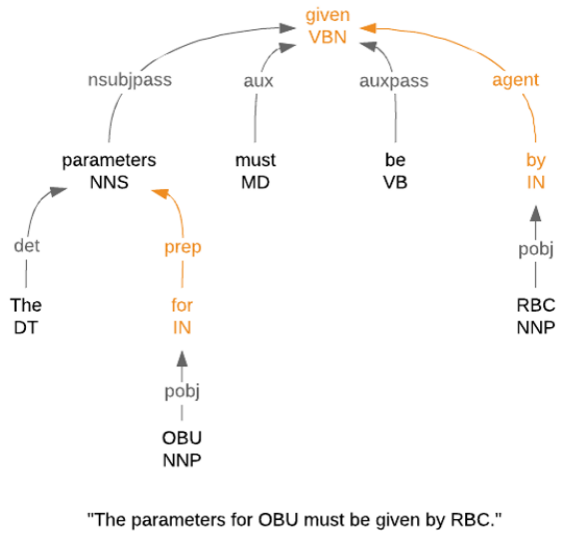


FIGURE 3.5: Detected branches of the tree in the first rule.

Second Level Rule

The second rule is defined as those dependency types of the descendant branches of one node in which their direct term has to be marked as a possible candidate if its PoS tag is relevant. After that, all the descendant branches of that child has to be analyzed by applying all the rules again.

Note that this rule is related to those patterns where the potential candidates are contained, such as subjects, object, noun compounds, etc. Additionally, in the same way as in the patterns table 3.1, this rule needs a PoS tag awareness as sometimes there are tags that are not relevant, so they have not to be taken into account. In addition, patterns shown in table 3.1 appear at the same time dependency types that are direct and non-direct to the candidate, and in the last case, these dependencies are taken into account in this second rule.

Table 3.3 shows the dependency types of the second rule, which can be easily reduced or expanded by adding or removing types of dependencies to the list.

Note that in table 3.3 there are dependency types of branches that do not appear

Dependency	Branch Type Definition
1	Nominal subject (NN)
2	Passive nominal subject (NN)
3	Noun compound modifier (NN)
4	Appositional modifier (NN)
5	Conjunct (NN)
6	Abbreviation modifier (NN)
7	Modifier in hyphenation (NN)
8	Numeric modifier (CD)
9	Direct object (any tag)
10	Indirect object (any tag)
11	Object of preposition (any tag)
12	ROOT (NN)

TABLE 3.3: Dependency type of branches of the second rule.

in the patterns of table 3.1 (specifically, they are abbreviation modifier, modifier in hyphenation and numeric modifier). This is because they do not appear in our study case, but we know that they are necessary to extract the whole meaning related to the ontology knowledge.

Moreover, "conjunct" dependency appears in both rule tables, as it can be a keyword if it is a name, or a redundant word in other cases, such as a preposition.

Figure 3.6 shows the branches of the dependency tree, which are marked with green color, that would be taken into account with the second rule. In this example, the "nsubjpass" dependency branch is taken into account by the dependency 2 of the second level rule table (table 3.3), and it can be seen in the pattern 1 of the patterns table (table 3.1). Then, the "pobj" dependency branch descending of the preposition branch is taken into account by the dependency 11 of the second level rule table, and it can be seen in the pattern 3 of the patterns table. Finally, the "pobj" dependency branch descending of the agent branch is taken into account by the dependency 11 of the second level rule table, and it can be seen in the pattern 4 of the patterns table.

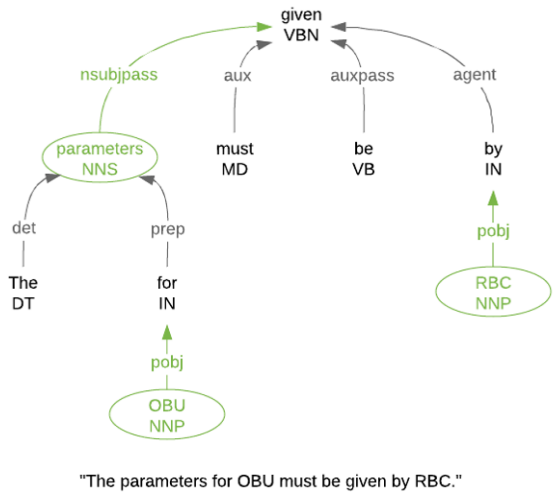


FIGURE 3.6: Detected branches of the tree in the second rule.

Third Level Rule

Finally, the last rule is the one that applies on the branches where none of the previous rules apply, which means that they are unknown branches. These branches are immediately pruned and neither the child terms nor its descendants will be analyzed.

Note that as we cannot be aware of all the dependency types that currently exist, because there are a large amounts of possibilities, the dependency types of branches in this rule is not defined as we avoid all ones that are not considered in the previous two rules.

Figure 3.7 shows the branches of the dependency tree, which are marked with red color, that are not analyzed. Note that, in that case, these types of branches do not contain any children; however, this is only a simple example phrase and it has to keep into account that there are more cases with this type of branches that we consider as not relevant.

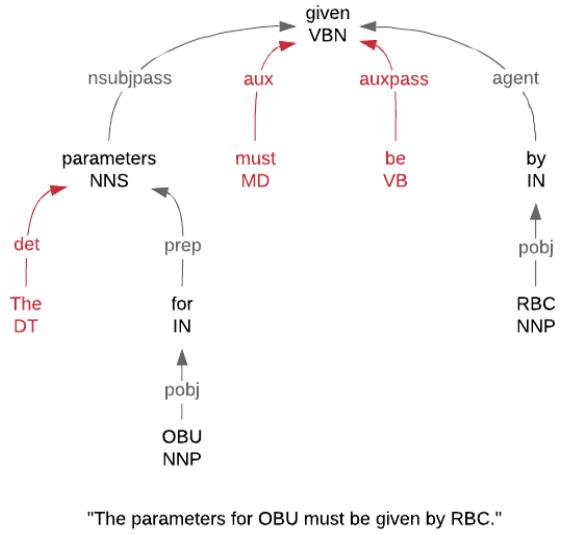


FIGURE 3.7: Detected branches of the tree in the third rule.

Example

Assuming all the rules explained in the previous paragraphs, an example of the path to extract the deepest keyword is analyzed and illustrated in figure 3.8.

Thence, knowing that the first state is the root node, all its descendant branches have to be analyzed by applying all the rules in order to find some candidate. In that case, the first descendant branch is the "nsubjpass" (dependency 2 of table 3.3 and pattern 1 of table 3.1), which will be detected in the second rule. Besides, as the PoS tag of its term is a noun, this word will be considered as a keyword of the sentence and will be marked as a candidate. Then, as the second rule states, the following step will be the analysis of the descendants of the "nsubjpass" node, where the first branch type is not considered as relevant in any of the first two rules, so the third one will be applied and that branch will not be analyzed. Then, the second branch of the node is detected in the first rule, so the descendants of that node will be analyzed without taking into account the term of the node, which in that case is a preposition (dependency 3 of table 3.2). Finally, the branch of the descendant of the preposition node is a "pobj" type (dependency 11 of table 3.3 and pattern 3 of table 3.1), so its term will be marked as a candidate.

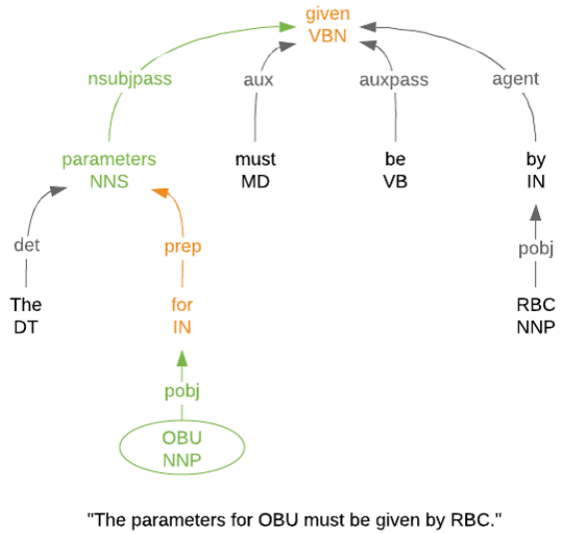


FIGURE 3.8: Deepest path for term extraction example.

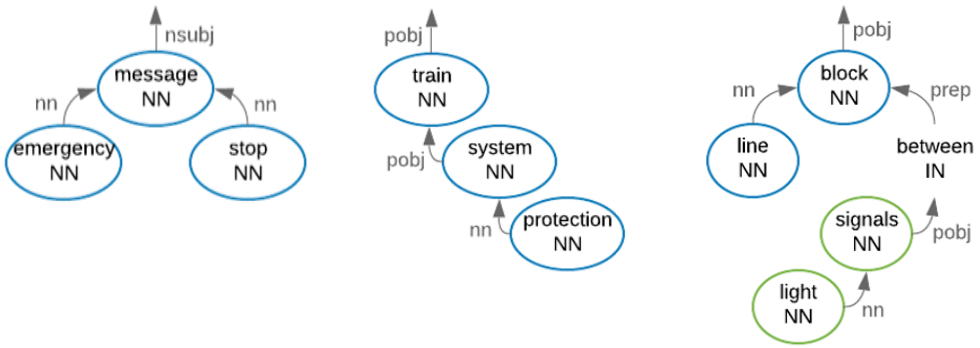
3.2.2.5 N-Grams Generation

N-grams generation is an essential step regarded to the IE process. Thus, once each branch of the tree was studied, and its terms were marked as word candidates, it is important to extract n-grams from those candidates that are directly connected within the tree, forming a set of words which encapsulate a big concept with a superior meaning to those single words. This step is highly important because ontology classes may contain concepts made with more than one term, so, in order to extract the correct meaning of the keywords within requirements, those n-grams have to be generated to avoid the loss of meaning and provide greater results of the dependency detection component.

Thus, this process merge those words with a parent-child relation generating a set of words that may contain one or more n-gram, where n is equal or greater than 1 and smaller or equal to the number of words in the set ($1 \leq n \leq set\ length$). This theorem includes different possibilities to generate a set of words (figure 3.9):

- The parent, which is a potential word candidate, has one or more relations with nodes that are also potential candidates, so all its terms are merged forming a set of words to be analyzed (figure 3.9a).
- The child of the relation already contains a set of words and the parent is also a potential candidate to be matched with the ontology, so the parent term is merged with the child generating a bigger set of words (figure 3.9b).
- The child of the relation is a potential candidate to be matched with the ontology, and it already contains a set of words from one of its children, but it is not directly connected with its parent, so the parent term is merged with the child word generating another set of words (figure 3.9c).

Assuming these situations, an important factor to recall, at the time of extract the n-grams, is the order of the words in the dependency tree, and as a consequence, in the set of words. Knowing that the words that form the concept in the original phrase and in the ontology class may be written with a different syntax order, that the parsing algorithm is non-projective (which means that the grammar structure



(A) A parent with two direct term children to merge in a unique set of words.

(B) A parent with one direct term child that has its own set of words to be merged in a unique set of words.

(C) A parent with one direct term child to merge in a set of words, and another separated set of words from the other not relevant child.

FIGURE 3.9: Different set of words situations to form n-grams.

of the dependencies are not syntactically ordered) and also that the potential candidates in the set of words that have been extracted may not appear in the ontology, it is important to understand that the extraction of the n-grams does not have to follow a typical sequence of an n-gram.

For example, lets assume that the original order of text in figure 3.9b is "train protection system", but the sequence of the dependency tree is "train system protection", hence, logical bi-grams such as "train system" and "system protection", which are extracted from the sequence order from the tree, may not be relevant in our ontology. Thence, the n-grams extraction has to find all the possibilities of the word combination to find a correct concept that can match with our ontology, such as "train protection".

For those reasons, all combinations of words are necessary to extract different n-grams that may match with the input ontology.

Finally, when all these syntax analysis steps are executed, the expected outcome will be a set of top concepts formed by several n-grams, where n is ($1 \leq n \leq$ *extracted top word set length*), as it is previously mentioned.

3.2.3 Semantic Analysis

Semantic analysis is the process that pretends to interpret the language meaning in order to understand which is the topic concept that comprehend the whole text. In other words, it allows us to extract the meanings of the most important words of the text, and also to transform them into comprehensive lemmas, obtaining the meaning of each word of the n-gram, and join them to get a unique meaning regarded to the concept of the ontology.

In this work, semantic analysis, which is based on lemmatization and semantic similarity, is used to extract features of the previous gathered keyword candidates in order to improve the results of the component and avoid ambiguities.

3.2.3.1 Lemmatization

Lemmatization is a semantic process used to obtain a particular unit form, also known as lemma, that involves the source meaning of the word. The framework DKPro is used as a support tool to apply the ClearNLP lemmatizer which is mentioned in previously steps of the approach (section 3.2.2.2). This particular component is based on a morphological analyzer which generates lemmas from given input tokens by the application of several rules with the support of a large dictionary gathered from various sources and several advanced heuristics.

In order to apply this process, it is necessary the knowledge of each PoS tag within the sentence to find the correct lemma of the word. The utilization of the tag is essential in the method because the same raw word can appear in the dictionary more than one time, and depending on its PoS tag it can have a different lemma. Furthermore, this particular lemmatizer also normalizes numbers, cardinals and redundant punctuation, among others.

It is important to recall that lemmatization is applied in each word of the gathered n-grams and to each term of the ontology classes. The reason of the application of that method to both data is to obtain the same form of the word (lemma) and to know if they are the same word even they are different words within the same lexeme, disambiguating the derived composition of the word to the same lemma.

Finally, these lemmas are saved as features in order to be analyzed in the ontology categorization step (section 3.2.4).

3.2.3.2 Semantic Similarity

Since requirements are written in NL, and we are not applying any quality process yet to avoid ambiguity of similar words or to constrain synonym words into a unique same word, semantic similarity metrics are needed to identify those similar words with the same meaning but with a different lemma. In this work, semantic similarity is used to detect synonyms between the extracted n-gram candidates and the ontology class concepts, both of them in their lemmas form, by the support of an input threshold which indicates the minimum similarity grade that stakeholders consider necessary to be synonyms in the ontology.

It is important to recall that this phase is also useful to accurate lemmatization step, as sometimes words that exists in the same lexeme, but they have specific PoS tags that in lemmatization step results in different lemmas, such as "interlock" and "interlocking", cause a false negative categorization, which is solved in this step, as they are classified as synonyms.

To deal with this process, the framework DKPro-Similarity is used. It provides several text similarity measures, algorithms and lexical-semantic resources. Concretely, this framework is used as word pair similarity detection where WordNet resource is used to extract similarity measures by a comparative algorithm regarded to that resource. This algorithm is a lexical semantic one which is known by the name Wu&Palmer (WuP) algorithm [38], which was chosen to be used in this step as it is simple to calculate and have good performances compared to the other similarity measures [39], and also is a more natural and direct way of evaluating semantic similarity in WordNet [40].

The principle of WuP algorithm is based on path/edge counting method of the ontology knowledge graph (WordNet). In other words, calculates the relatedness by considering the depths of the two synsets in the WordNet taxonomies, along with the depth of the least common subsumer (LCS) of the two concepts, which

is known as the most specific concept that is an ancestor of both terms within the WordNet ontology. Thence, the similarity measure of WuP algorithm is defined by the following expression [41]:

$$Sim_{WuP}(w_1, w_2) = \frac{2 * depth_{LCS}}{depth_{w_1} + depth_{w_2}}; \qquad where\ 0 \leq Sim_{WuP} \leq 1 \tag{3.1}$$

Assuming that, in equation 3.1, the depth function is the measure of the shortest edge counting of the WordNet taxonomy graph from the ROOT until the indicated word, and the depth for the LCS is the shortest edge counting from the ROOT until a common ancestor of both terms.

It is important to recall that this process is time-expensive and it has to be run several times to get the similarity between requirement extracted terms and ontology class concepts. Thence, in order to optimize the outcome timing, instead of generating the similarity measures features of all the candidates to be matched in the ontology, this step is run at the same time of the ontology categorization (section 3.2.4) for only those n-gram candidates that can not be categorized in the ontology (i.e., not direct match between the lemmas of the candidates and the ontology concepts has been found) and requires a synonym analysis, improving the precision of this work.

Additionally, as several words analyzed in this phase are technical and are constantly analyzed since they appear many times throughout the document, in order to reduce the run time of the component execution, a learning step is developed, where synonymy of pairs of words are saved to learn its similarity in future requirement analysis of the document.

3.2.4 Ontology Categorization

Categorization is a process in which data objects are identified, differentiated and understood to be grouped into several categories, classes or clusters. In our case, we use categorization to group input requirements into the different concept classes of the ontology by their similar features, such as raw words, lemmas and semantic similarity measure. Particularly, the approach of *conceptual clustering* is used to achieve this goal.

Thence, conceptual clustering is a machine learning process which arranges clusters around predefined concept classes by the features extracted with the utilization of several semantic techniques and resources, such as WordNet and Wikipedia concepts, in order to classify and understand the input data according to each class concept or description [42], [43].

Knowing that definition and how conceptual clustering works, due to the reason that in this work each cluster is identified as an ontology class which contains a particular concept formed by an n-gram, which usually is a uni-gram, bi-gram or even a tri-gram, the previous steps of this approach have been oriented to the extraction of relevant information for this categorization of concepts where requirements can be clustered around the classes of the provided ontology, allowing to know its dependencies.

Thence, extracted features from requirements, such as gathered n-grams from IE process, their lemmas forms obtained from lemmatization and similarity measures from synonyms calculated in the semantic similarity process, are examined in order to find the same combination of features in the concepts of the ontology.

Additionally, as a requirement can contain more than one potential candidate, and as it can also be formed by sub-requirements that could talk about more than one known concept of the ontology, an adaptation of this method is required in order to identify all the dependencies of a requirement. Hence, one requirement may belong to one or more groups of the ontology and not only to one cluster.

Therefore, the method to analyze these features and cluster each requirement into the previous defined concepts follows a sequential process for all extracted n-grams from each requirement and for all ontology classes:

1. Identify whether the extracted n-gram from the requirement is the same, or a combination of words, as the n-gram from a concept ontology class. In case of finding an equivalent feature it will be marked as individual of that class.

2. If the previous condition is not satisfied, identify if the extracted lemma of each word within the n-gram from the requirement is the same, or a combination of words, as the lemma of each word within the n-gram from the concept ontology class. In case of finding an equivalent feature it will be marked as individual of that class.

3. If both previous conditions are not satisfied, calculate the semantic relatedness between the lemmas from the n-grams of both requirements and ontology classes. If the similarity measure is bigger or equal than the provided threshold, requirement will be marked as individual of that class.

4. If none of the previous clauses are satisfied, it is not considered that this particular requirement has dependencies, so it is not saved as an individual in the ontology.

Once all these steps are executed, the ontology will be full of requirement individuals.

3.2.5 Dependency Extraction

The identification of each defined dependency within the ontology between the provided requirements is the last step of this approach. It lies on the dependency link detection between ontology classes and it is the reason why this approach is based on an ontological knowledge.

Thence, each ontology class is analyzed in search of individuals instantiated in that class, and also individuals instantiated in the classes that are related to the class analyzed with a dependency link, in other words, two classes that are linked with a dependency relation within the ontology are analyzed extracting its instances to know the dependencies between each requirement.

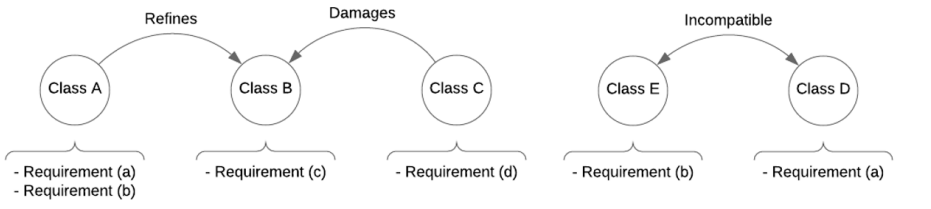


FIGURE 3.10: Dependency link identification example.

Figure 3.10 shows a basic example of an ontology once it is fulled with their individual requirements for each class. In that example, all requirements from class

A refines all requirements from class B, which damages all requirements of class C. Moreover, as it is seen, requirements (a) and (b) have also matched with classes D and E respectively, which have a dependency of incompatibility between them, meaning that they can not exist at the same time in the project.

Note that the ontology is the knowledge where dependencies are defined, so any modification, such as addition or deletion, of dependencies or classes in the ontology means a variation in the outcomes of this component.

Chapter 4

Evaluation and Discussion

In order to know the degree of efficiency and validate the correctness of all the functionalities of the component implemented in this work, the evaluation and discussion of the results are illustrated in this chapter. Additionally, a brief explanation of the dataset used to test and validate the component, and the provided ontology to extract the requirements dependencies are also exposed.

4.1 Dataset

In order to know the complexity to which the software is confronted and to comprehend the results of the component, it is important to understand where the data comes from, what their professional domain is, how they are structured and what their features are. Thus, in this section, the data is analyzed in order to understand their context and their characteristics, and to comprehend how the outcomes of the proposed approach are therefor.

As it was previously mentioned, the data available to evaluate this software is provided by one of the industrial partners of OpenReq, concretely by Siemens. This data is formed by several documents called Request for Proposals (RFP) on the railway domain. The documents comprise natural language requirements and can be several hundred pages long. Being specific on the railway domain, each requirement may contain technical words that are also part of the domain-specific ontology.

The main goal of Siemens being involved in the OpenReq project is to exploit OpenReq as a support to find a good solution to cover all requirements and to reduce costs of both the proposal phase and the requirements analysis phase of projects. Among the different OpenReq's functionalities they are interested in to achieve this goal, the automated dependency management is one of them.

Moreover, in each RFP provided by Siemens we can find requirements that deal with electronic interlocking installations, automated line blocks, building materials, software, reconstructions, rehabilitation or extension of tracks and many other topics related to railways.

4.1.1 Data Analysis

The provided data consists in six RFP which are written in NL and contains both requirements and non-requirements clauses. It is important to recall that those documents are processed by a prior component of the OpenReq project, which is in charge of extracting all requirements within them, and classify whether those clauses are requirements or not. Thus, the input data that our component requires is only a set of requirement clauses which are previously extracted by that OpenReq component.

Table 4.1 shows the set of provided RFPs and the total requirements that we have to analyze to extract their dependencies:

Trial	Total clauses	Requirements
RFP 1	1525	1209
RFP 2	8123	6880
RFP 3	1854	1535
RFP 4	1382	1140
RFP 5	853	746
RFP 6	3819	3204

TABLE 4.1: Set of documents provided by Siemens

Nevertheless, due to the complexity of these type of documents written in NL, where a large amount of technical terms are specified, the provided requirements are split by pieces of texts or paragraphs (where each one of them may contain more than one sentence), and they do not follow any requirement template. Thence, requirements that we own are not only requirements in themselves, but everything that surrounds them, such as connectors, expressions, phrases within parenthesis, etc. In addition, we can also find bullet lists where perhaps each point does not reach to make a complete sentence, but a sequence of properties of the requirement.

Thence, the data that we have to deal with is a part of text containing a requirement and its surroundings, or even a part of text that was detected as requirement because of its situation in the document, but does not follow a typical requirement template. So, we can say that a quality process has not been applied to the provided data, generating a possible problem of ambiguity that can reach to a complication for the outcome accuracy, and that this approach has to deal with.

4.1.2 Input Ontology

As it is thoroughly mentioned in the section 3.1.3, the dependency detection component developed in this work requires an input ontology to deal with all the dependency semantic relations between requirements. Knowing that, Siemens has provided us with a specialized ontology for the domain of the supplied data, which follows a previously designed structure for this component.

Therefore, as it is known, each ontology class is a concept that has a dependency to another ontology class, creating a network of knowledge relations.

Figure 4.1 shows the provided ontology, where most of the concepts are specific technical terms used in railway mobility domain or general terms which are also related to that kind of project domains. In this particular case, ontology classes follow a hierarchical structure, which is indicated by the blue links that are not dependency relations. Differently, the other links represent diverse dependencies. Particularly, in this ontology there are only three type of dependencies: "refines", "requires" and "incompatible". Thus, the orange links symbolize the relation called "refines" (e.g., ETCSLevel1 refines ETCS). Otherwise, yellow links represents the dependency named "requires" (e.g., RailwayStation requires RailwaySwitch). And finally, the third one is a brown link, which is only used twice in this ontology, and it is used for "incompatible" relations (e.g., TrackCircuit is incompatible with CountingPoint).

Knowing that, the test for our approach has to categorize the provided requirements into those classes in order to extract the dependencies defined in this ontology.

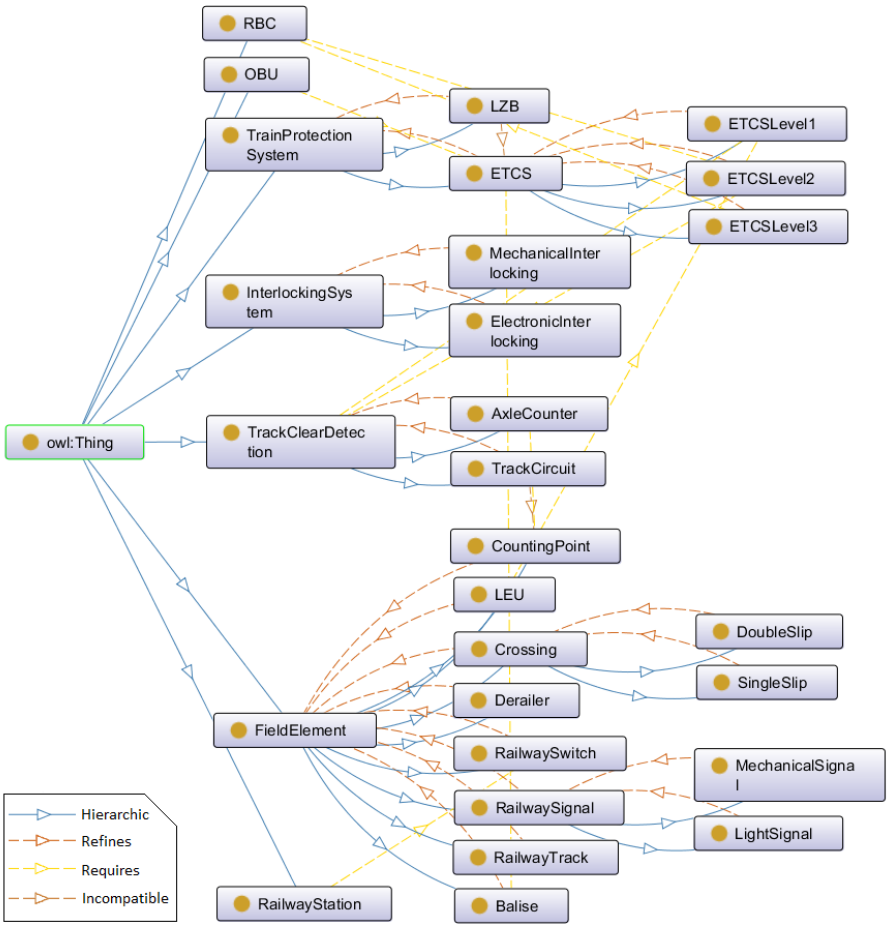


FIGURE 4.1: Railway domain ontology graph.

4.2 Dependency Detection Results

Final outcomes of the dependency detection component are shown in this section in order to evaluate its efficiency, validate its functionalities and see if the component achieve the objectives proposed at the beginning of the project.

Accordingly, to be able to make a correct evaluation of the outcomes, several tests have been run based on the provided ontological knowledge and on all the RFPs of the set of documents previously mentioned, obtaining the number of dependencies shown in the table 4.2.

Thence, knowing that this type of requirements usually have several dependencies of different semantic types, a large number of them have been extracted from each RFP. Thus, as it is expected on the results, the number of detected dependencies is not proportional to the number of requirements, but varies according to the document, which may contain a bigger or smaller amount of interdependencies depending on its own content, as can be seen in RFP 4 that contains less dependencies than RFP 1, where there are more dependencies than requirements, even though the number of their requirements is similar.

Analyzing these results, it can be assumed that the information extraction with the support of a dependency parser can achieve the identification of potential candidates and matches them into the ontology to extract its dependencies.

In addition, to prove the efficiency of the semantic analysis phase, some lemmas obtained in the lemmatization step that matches with the ontology, and some similar

Trial	Requirements	Dependencies	Dependencies (without synonymy)
RPF 1	1209	4453	4097
RPF 2	6880	12242	9987
RPF 3	1535	182	182
RPF 4	1140	167	72
RPF 5	746	39	39
RPF 6	3204	1261	753
Total	**14714**	**18344**	**15130**

TABLE 4.2: Detected dependencies within the provided RFPs.

requirement lemmas detected as synonyms of the ontology lemmas are extracted. Thence, as it is seen in table 4.3, words such as "crossings" and "RBCs" match with the ontology terms with the support of their lemmas, as their original forms are not properly composed to be matched with the ontology. Otherwise, other words such as "interlock" does not match with the ontology term but match with the ontology lemma.

Requirement Term	Requirement Lemma	Ontology Term	Ontology Lemma
systems	system	system	system
crossings	crossing	crossing	cross
RBCs	rbc	RBC	rbc
switches	switch	switch	switch
interlock	interlock	interlocking	interlock
railroad	railroad	railway	railway
signaling	signaling	signal	signal
rail	rail	track	track
mark	mark	cross	cross
track	track	cross	cross

TABLE 4.3: Subset of requirement's terms and lemmas that matches with ontology concepts.

Furthermore, we detect useful lemmas classified as semantically similar (synonymy) that improve the accuracy of our results, such as "railway" and "railroad", or "signal" and "signaling", among others.

Nevertheless, during the analysis of the results, despite we detect that useful terms classified as semantically similar, it has also been detected some other terms identified as semantically similar that can result in a wrong similar detection if the concepts of the ontology classes are ambiguous. This is the case of the concept *Crossing* from the provided ontology, which lemma used in the WuP similarity algorithm (section 3.2.3.2) is "cross" and it can be detected as synonym of numerous terms that are not related with the concept of the class, such as "mark", "cover" or "track", among others, concluding in wrong results in the requirements categorization phase.

Due to these ambiguous categorizations, in table 4.2, we also shown the tests for each RFP without the semantic similarity method in order to see the importance of the similarity phase and to know if it is a crucial step in future works related to this project. Thus, as it is seen, the component without the semantic similarity phase is still detecting a huge number of dependencies, so it can be affirmed that these phase

is only a refinement step to improve the accuracy of our results, where in order to solve the synonymy misunderstanding, it is important to ensure the disambiguation of all concepts within the input ontology. In order to fix this problem, it is just needed to replace the ambiguous term *Crossing* with the unambiguous concept *Level Crossing*, as it is a more technical expression within the railway domain, as it was detailed by Siemens' experts.

4.2.1 Validation of the Results by Experts

Due to no labels of dependencies are given to verify the results of the dependency detection algorithm, we can not fill the confusion matrix with the predicted results in order to calculate the accuracy of the component. Thus, in order to validate our results, Siemens' experts made a manual validation of the detected dependencies outcomes of our component in order to know whether the predicted observations are correct or not.

Then, this validation is important to know the precision of the approach, and to be aware if the objectives of the project have been achieved. Thence, Siemens' experts made an evaluation of a random subset of requirements extracted from three different RFP, where around 200 dependencies were manually analyzed and validated.

Table 4.4 shows a statistical calculation of the validated results, also known as precision measure, which is the ratio of correctly predicted positive observations among the total predicted positive observations. However, due to the complexity and ambiguity that a requirement may entail, three precision degrees have been categorized in order to go in depth in the evaluation and discussion of these results. These three statistical degrees are related to the precision of true positive detected dependencies, the precision with a refinement possibility (Precision-R) of true positive detected dependencies, and the imprecision of the detected dependencies, which is related to false positives outcomes.

Precision	**Precision-R**	**Imprecision**
89.2 %	7.7 %	3.1 %

TABLE 4.4: Precision measure of validated outcomes.

Therefore, precision (89.2%) takes into account those dependencies that do not contain any errors, either in the semantics of the dependency or in the cluster in which both requirements have been categorized, and classify them as correct dependencies. This high percentage shows the efficiency of the component.

In table 4.5, some examples of these correct dependencies are shown, where the first column represents the requirement where the dependency comes from, the second column represents the requirement where the dependency goes to, and the third column contains the dependency type between both requirements and its respective ontology concepts, which are extracted within each requirement to identify that particular dependency.

An important aspect to recall in those dependency examples is that the three of them are composed by the same requirement, either it is the one where the dependency comes from or where it goes to, which contains three different concepts defined in the ontology and detected in the approach, such as *Balise*, *ETCS Level 2* and *RBC*, which makes three dependencies with three other requirements that contains other concepts specified in the provided ontology, such as *Field Element*. In the

examples, the dependencies are formed by two different semantics, which are "refines" and "requires". Thus, it is proved that the approach can deal with more than one semantic dependency and detects several concepts defined in the ontology, even if they are in the same requirement.

From	To	Dependency Type
RBC 320 All acceptance tests for the ETCS level 2 system shall check the content of the telegrams emitted/received by the RBC and the balises.(LC)	B. Modules: Modules are considered replaceable parts of interfaces with field elements. The bidder can however adapt this according to his equipment concept.	[Balise] refines [FieldElement]
RBC 320 All acceptance tests for the ETCS level 2 system shall check the content of the telegrams emitted/received by the RBC and the balises.(LC)	RBC 198 Shunting routes need not be recorded in the RBC.(LC)	[ETCSLevel2] requires [RBC]
CE 3. The existing centralization technology based on the relay technology that equips project stations and the related BLA distances of the Employer must be rehabilitated to be compatible with ERTMS with ETCS level 2.	RBC 320 All acceptance tests for the ETCS level 2 system shall check the content of the telegrams emitted/received by the RBC and the balises.(LC)	[ETCSLevel2] requires [RBC]

TABLE 4.5: True positive dependency examples.

Dependencies labeled in Precision-R (7.7%) are those that are correct due to the categorization logic that has been followed in the approach, but where a human can refine that categorization by understanding some ambiguous meanings within the requirement clause. In our validation tests, only one case was detected.

In table 4.6, some examples of those dependencies are shown. In this particular case, the ambiguous concept that the approach has to deal with is "ETCS / ERTMS Eurobalises". Thus, as it is expected, the approach detects the term "ETCS" and categorizes that concept in its properly ontology class, whose outcome is correct for our approach. However, after these dependencies were analyzed by Siemens' experts, we can ensure that this case is ambiguous, as they suggest that the concepts "ETCS" and "ERTMS" are only the prefixes of "Eurobalises", which is the principal idea of the requirement. This particular concept is related to the class *Balise* of our ontology, but it is not detected as lemma or semantically similar to it because is a technical term that is not specified in the used semantic models and dictionaries, so it is not properly detected causing a misunderstanding and ambiguous categorization.

Thus, due to the categorization of the requirement into *ETCS* ontology class is not incorrect, but a misunderstanding or a secondary dependency, and the mentioned concept (Eurobalise) needs a refinement to be detected as the principal idea of the requirement, the suggested point to solve this ambiguity would be the addition of the concept within some glossary that allows its detection in the semantic analysis phase, or the addition of the concept within the ontology as a depth refinement class of *Balises*. This point is part of the future work of this thesis.

From	To	Dependency Type
The level ETCS/ERTMS level 2 system requires the implementation of a dedicated video surveillance system. To ensure high-level video information, the areas where there are point machines and level crossings as well as the persons and all locations required by the beneficiary will be monitored. The following works will be executed:	installation of ETCS/ERTMS Eurobalises;	[ETCSLevel2] refines [ETCS]
The level ETCS/ERTMS level 2 system requires the implementation of a dedicated video surveillance system. To ensure high-level video information, the areas where there are point machines and level crossings as well as the persons and all locations required by the beneficiary will be monitored. The following works will be executed:	the installation of the ETCS equipment (eurobalise) for all lines in the station and on the IABS installation.	[ETCSLevel2] refines [ETCS]

TABLE 4.6: True positive dependency examples to be refined.

Ultimately, imprecision (3.1%) covers the dependencies that have been detected as incorrect, since they are not valid due to some ambiguity in the text or ontology. Some examples of false positives are shown in table 4.7.

From	To	Dependency Type
the contrary track signals must be stopped.	system interfaces with field elements shall include a minimum of relays; communication with field object interfaces is safe and reliable;	[Crossing] refines [FieldElement]
After work completion, the contractor shall hand over to the beneficiary a drawing with the current status of cable laying, indicating the position (kilometre) of the junction thimbles. These shall be marked on the field by means of thimble marks.	• Communication with field elements interfaces must be safe and reliable.	[Crossing] refines [FieldElement]
implementation of the Level 2 ETCS System. The system consists of:	The level ETCS/ERTMS level 2 system requires the implementation of a dedicated video surveillance system. To ensure high-level video information, the areas where there are point machines and level crossings as well as the persons and all locations required by the beneficiary will be monitored. The following works will be executed:	[ETCSLevel2] refines [ETCS]

TABLE 4.7: False positive dependency examples.

Nevertheless, only two types of incorrect dependencies have been detected. The first one is previously mentioned in this chapter (table 4.3), where some lemmas are wrongly detected as synonym due to some ambiguous term in the ontology, such as *Crossing*, concluding in categorization errors (e.g., "cross" and "track", "cross" and "mark"). The second one is the case where an ontology class concept is involved in the concepts of its children classes (e.g., *ETCS* and *ETCS Level 2*), causing a detection of both concepts as candidates if "ETCS Level 2" is contained in the requirement clause, which is not a proper categorization of the "ETCS" term, but is absolutely correct for "ETCS Level 2" tri-gram.

Thus, in order to control these detected errors, two points are suggested as future work of this thesis. For the synonym case, it is needed a disambiguation of the ontology concept, or on its absence, a learning process where ambiguous synonym relations will be debugged and validated avoiding misunderstandings. On the other hand, the wrong categorization of a parent-child relation case should be controlled by the single categorization in the most specific class (child) of that ontology relation.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Motivated by the problem of automatically detecting dependencies on a semantic level to improve the usually manual, laborious and inefficient methodology to manage dependencies, the research and development of this work has taken us through a long path of exploring the traceability and management of requirements information within the RE field, the investigation of its state-of-the-art models and applications, the understanding of the importance of the quality and well-written NL requirements, the awareness of the gap in the detection of several dependencies to monitor the coherency of requirements within the same request for proposal, the need to develop a new model to fill that gap, and the successful application of this new model to achieve the objectives of this work.

Artificial Intelligence algorithms seem to be a recurrent approach to deal with NL requirements and extract information to manage dependencies between them. A survey of previous articles shows us that many models have been proposed to achieve that problem related to dependencies. However, we have not found approaches that deal with more dependency types than incompatibility, as this work aims to perform. These approaches, though, inspired the proposal of a new dependency detection model.

Thence, having domain knowledge about that previous models that proved to be useful to achieve a simpler dependency identification, in this work we presented the approach to the automatic identification of dependencies for NL requirements on a semantic level. The approach aims to detect different types of dependencies, such as refines, requires and incompatible, among others, by the combination of several techniques of NLP and ML, such as data preprocessing, dependency parsing, information extraction and conceptual clustering, together with an ontology knowledge that provides to the component the different dependency definitions between term concepts involved in a specific domain, which is the core of the component. The key part of this approach has been the identification of that words within a single requirement that are pondered as the keywords of its clause and have to be matched with the predefined semantic concepts of the ontology.

The evaluation results detailed in chapter 4 show the successful application of the approach to a real-world use-case dataset, which was provided by Siemens. The validation of the results presents a precision of 89.2% of correct dependencies, a precision of 7.7% of correct dependencies that can be improved by disambiguation, and an imprecision of 3.1%. These percentages show the efficiency of the component. Additionally, the results show an improvement of requirement categorization by the utilization of the semantic analysis phase, where lemmatization and semantic similarity has been applied.

Therefore, regarding the sub-goals detailed at the beginning of this work, we can affirm that all of them have been successfully achieved for all the reasons concluded in the previous paragraphs, as it takes inspiration from other previous studies related on automatic dependency detection, applies an intelligent approach based on artificial intelligent algorithms (i.e., NLP and ML algorithms), and determines dependencies between requirements on a semantic level by the detection of several types of semantic dependencies.

Finally, since we have achieve these sub-goals, we can conclude a satisfactory achievement of the main objective of this work, which is *the development of a new approach to automatically detect dependencies between requirements* is successfully done and with great results.

5.2 Future Work

As is usual in a new proposed approach, there are several points to remark as a future work in order to improve the results on the researched field.

The first point, as mentioned in chapter 4, is to improve the percentage of correct dependencies of the results that need to be refined by a disambiguation process. In that case, the suggested point to implement in a future work is an addition of those technical concepts that can be misunderstood within some glossary that allows its clarification in the semantic analysis phase, or the addition of those concepts within the ontology as a depth refinement class.

The second point in the future work aims to reduce the imprecision percentage. There are two cases involved in these imprecise results. The first one is the case when an ontology concept is too general, causing misunderstandings and ambiguities. In order to solve that problem, our proposed solution is a disambiguation of the ontology concept by replacing it with a more specified one, or on its absence, a learning process where ambiguous synonym relations will be debugged and validated avoiding misunderstandings by the training of several pairs of terms that are synonyms or ambiguous. On the other and, the second case is detected on those ontology classes that the concept of a parent class is also found in the more specialized child concept (e.g., *ETCS* concept is also found in, *ETCS Level 2* concept), causing a duplicate requirement in the parent class (e.g., *ETCS*) instead of in only the specialized child one (e.g., *ETCS Level 2*). Our proposed solution controls the error by the single categorization in the most specific class (child) of that ontology relation.

Regarding the future research of this work, it would be interesting to develop a model that can automatically create an ontology that covers all technical terms of a specialized domain instead of having to generate it manually by experts. Additionally, with the support of a labeled data set of dependencies, which we have not been able to achieve, a supervised approach to train the IE phase would be a great optimization to improve the categorization and reduce the false negatives that could appear in this type of complex data. Moreover, it is needed to explore in a broader way what other type of dependencies can be identified with the models developed in this work.

References

[1] K. Pohl, "The three dimensions of requirements engineering: A framework and its applications", *Selected Papers from the Fifth International Conference on Advanced Information Systems Engineering*, CAISE '93, pp. 243–258, 1994.

[2] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem", *Proceedings of IEEE International Conference on Requirements Engineering*, pp. 94–101, Apr. 1994.

[3] E. Bouillon, P. Mäder, and I. Philippow, "A survey on usage scenarios for requirements traceability in practice", in *Requirements Engineering: Foundation for Software Quality*, J. Doerr and A. L. Opdahl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 158–173, ISBN: 978-3-642-37422-7.

[4] C. Ziftci and I. Krüger, "Getting more from requirements traceability: Requirements testing progress", *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pp. 12–18, May 2013.

[5] L. Lehtola, M. Kauppinen, and S. Kujala, "Requirements prioritization challenges in practice", in *Product Focused Software Process Improvement*, F. Bomarius and H. Iida, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 497–508, ISBN: 978-3-540-24659-6.

[6] A. Vogelsang and S. Fuhrmann, "Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study", *2013 21st IEEE International Requirements Engineering Conference (RE)*, pp. 267–272, 2013.

[7] H. Zhang, J. Li, L. Zhu, R. Jeffery, Y. Liu, Q. Wang, and M. Li, "Investigating dependencies in software requirements for change propagation analysis", *Inf. Softw. Technol.*, vol. 56, no. 1, pp. 40–53, Jan. 2014, ISSN: 0950-5849.

[8] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. och Dag, "An industrial survey of requirements interdependencies in software product release planning", *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pp. 84–91, Aug. 2001.

[9] H. F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects", *IEEE Software*, vol. 18, no. 4, pp. 58–66, Jul. 2001.

[10] A. P. Nikora and G. Balcom, "Automated identification of ltl patterns in natural language requirements", *2009 20th International Symposium on Software Reliability Engineering*, pp. 185–194, Nov. 2009.

[11] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions", in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. Hyderabad, India: ACM, 2014, pp. 55–69, ISBN: 978-1-4503-2865-4.

[12] OpenReq, *Project*, Accessed: 2018-08-25. [Online]. Available: https://openreq.eu/.

[13] X. Zhu and Z. Jin, "Inconsistency measurement of software requirements specifications: An ontology-based approach", *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 402–410, 2005.

[14] ——, "Ontology-based inconsistency management of software requirements specifications", *31st Conference on Current Trends in Theory and Practice of Computer Science. Lecture Notes in Computer Science, Springer Verlag*, vol. 3381, pp. 340–349, Jan. 2005.

[15] L. Li, B. Qin, and T. Liu, "Contradiction detection with contradiction-specific word embedding", *Algorithms*, vol. 10, no. 2, 2017.

[16] J. Misra, "Terminological inconsistency analysis of natural language requirements", *Information and Software Technology*, vol. 74, pp. 183–193, 2016.

[17] T. Moser, D. Winkler, M. Heindl, and S. Biffl, "Requirements management with semantic technology: An empirical study on automated requirements categorization and conflict analysis", in *Advanced Information Systems Engineering*, H. Mouratidis and C. Rolland, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–17.

[18] M. Temizer and B. Diri, "Automatic subject-object-verb relation extraction", *2012 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 1–4, 2012.

[19] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, and D. Mladenić, "Triplet extraction from sentences", 2007.

[20] M. Kim, S. Park, V. Sugumaran, and H. Yang, "Managing requirements conflicts in software product lines: A goal and scenario based approach", *Data and Knowledge Engineering*, vol. 61, no. 3, pp. 417–432, 2007, Advances on Natural Language Processing.

[21] A. Fantechi and E. Spinicci, "A content analysis technique for inconsistency detection in software requirements documents", *WER 2005*, pp. 245–256, 2005.

[22] C. Liu, "Ontology-based conflict analysis method in non-functional requirements", *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, pp. 491–496, Aug. 2010.

[23] K. Verma and A. Kass, "Requirements analysis tool: A tool for automatically analyzing software requirements documents", in *The Semantic Web - ISWC 2008*, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 751–763.

[24] M.-C. de Marneffe, A. N. Rafferty, and C. D. Manning, "Finding contradictions in text", *Association for Computational Linguistics-Human Language Technologies (ACL-HLT)*, 2008.

[25] C. Kulathunga and D. D. Karunaratne, "An ontology-based and domain specific clustering methodology for financial documents", *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 1–8, Sep. 2017.

[26] R. Navigli and M. Lapata, "An experimental study of graph connectivity for unsupervised word sense disambiguation", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 4, pp. 678–692, Apr. 2010.

[27] S. Tiun, H. Zakr, M. Mohd, N. Z. Abidin, and A. I. I. Hisham, "Word sense disambiguation for english quranic ir system", *2013 Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences*, pp. 199–202, Dec. 2013.

[28] D. Ustalov, D. Teslenko, A. Panchenko, and M. Chernoskutov, "Mnogoznal: An unsupervised system for word sense disambiguation", *2017 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, pp. 147–150, Sep. 2017.

[29] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation", *Machine Learning Research 3*, pp. 993–1022, 2003.

[30] S. Zhang, Y. Sha, and X. Wang, "Reviews analysis based on sentence and word relevance", *2014 Seventh International Symposium on Computational Intelligence and Design, Hangzhou*, pp. 43–46, 2014.

[31] G. Balikas, M.-R. Amini, and M. Clausel, "On a topic model for sentences", *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 921–924, 2016.

[32] K. Pohl and C. Rupp, "Requirements engineering fundamentals", *1st ed, Rocky Nook*, 2011.

[33] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)", *2009 17th IEEE International Requirements Engineering Conference*, pp. 317–322, Aug. 2009.

[34] A. OpenNLP, *Apache opennlp developer documentation*, Accessed: 2018-09-09. [Online]. Available: https://opennlp.apache.org/docs/1.5.3/manual/opennlp.html.

[35] J. D. Choi and M. Palmer, "Fast and robust part-of-speech tagging using dynamic model selection", in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12)*. Jeju, Korea, 2012, pp. 363–367.

[36] J. Nivre, "Algorithms for deterministic incremental dependency parsing", *Comput. Linguist.*, vol. 34, no. 4, pp. 513–553, Dec. 2008, ISSN: 0891-2017.

[37] J. D. Choi and M. Palmer, "Getting the most out of transition-based dependency parsing", in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers*, ser. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, vol. 2, pp. 687–692.

[38] Z. Wu and M. Palmer, "Verbs semantics and lexical selection", in *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ser. ACL '94. Las Cruces, New Mexico: Association for Computational Linguistics, 1994, pp. 133–138.

[39] T. Slimani, B. Ben Yaghlane, and K. Mellouli, "A new similarity measure based on edge counting", Jan. 2004.

[40] Y. Jin, H. Zhou, H. Yang, Y. Shen, Z. Xie, Y. Yu, and F. Hang, "An approach to measuring semantic similarity and relatedness between concepts in an ontology", in *2017 23rd International Conference on Automation and Computing (ICAC)*. Sep. 2017, pp. 1–6.

[41] V. Ganesan, R. Swaminathan, and M. Thenmozhi, "Similarity measure based on edge counting using ontology", *International Journal of Engineering Research and Development*, vol. 3, no. 3, pp. 40–44, Aug. 2012.

[42] A. Boubacar and Z. Niu, "Conceptual clustering", in *Future Information Technology*, J. J. Park, Y. Pan, C.-S. Kim, and Y. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–8.

[43] R. S. Michalski and R. E. Stepp, "Learning from observation: Conceptual clustering", in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 331–363.

Appendix A

API Documentation

In this appendix, the API documentation of the Dependency Detection component developed in this work is detailed:

Service	Dependency Detection
URL	http://217.172.12.199:9407/upc/dependency-detection/json/ontology/{projectId}
Explanation	Uploads an ontology (in RDF/XML language) and a JSON object to the server, extracts the dependencies (using the ontology) of the requirements of the project which id is sent by parameter and finally removes the uploaded file.
Method	POST
URL Parameters	projectId: Id of the project where the requirements to analize are: - Parameter Type: path - Data Type: String

JSON Body	json: The json object to upload. Follows the OpenReq JSON format.
	- Parameter Type: formData
	- Data Type: String
	
	Example:
	{
	"projects" : [
	{
	"id" : "DD",
	"name" : "Dependency detection",
	"specifiedRequirements" : ["A", "B"]
	}],
	"requirements" : [
	{
	"id" : "A",
	"name" : "a",
	"text" : "The system should use Mahout."
	}, {
	"id" : "B",
	"name" : "b",
	"text" : "The system should use Java."
	}],
	"dependencies" : [
	{
	"dependency_type": "requires",
	"status": "proposed",
	"fromid": "A",
	"toid": "B",
	"component" : "dependency-detection"
	}]
	}

Ontology Body	ontology: The Ontology file to upload (RDF/XML lang.) - Parameter Type: formData - Data Type: file Example: <?xml version="1.0"?> <rdf:RDF xmlns="http://www.semanticweb.org/openreq/ demo#" xml:base="http://www.semanticweb.org/openreq/demo" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns# "xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:xml ="http://www.w3.org/XML/1998/namespace" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"> <owl:Ontology rdf:about="http://www.semanticweb.org/ openreq/demo"/> <owl:ObjectProperty rdf:about= "http://www.semanticweb.org/openreq/demo#requires"/> </owl:DatatypeProperty> <owl:DatatypeProperty rdf:about= "http://www.semanticweb.org/openreq/demo#id"> </owl:DatatypeProperty> </owl:DatatypeProperty> <owl:DatatypeProperty rdf:about= "http://www.semanticweb.org/openreq/demo #requirement"> </owl:DatatypeProperty> <owl:Class rdf:about= "http://www.semanticweb.org/openreq/demo#mahout"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource= "http://www.semanticweb.org/openreq/demo#requires"/> <owl:someValuesFrom rdf:resource= "http://www.semanticweb.org/openreq/demo#java"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class> <owl:Class rdf:about= "http://www.semanticweb.org/openreq/demo#java"> </owl:Class> </rdf:RDF>

Returned Data	Follows the OpenReq JSON format.
	
	Adds "dependency-type" objects to the "dependencies" array, such as:
	{
	"dependency_type":"requires",
	"fromid":"B",
	"toid":"A",
	"status":"proposed"
	"component": "Dependency-detection"
	}

Appendix B

Glossary

In this appendix, three glossaries regarded to the acronyms, Part-of-Speech tags and dependency abbreviations from the dependency parser used during the development of this work are detailed.

B.1 Glossary of General Acronyms

Acronym	**Definition**
ANN	Artificial Neural Network
EU	European Union
ICT	Information and Communications Technology
ID	Inconsistency Detection
IE	Information Extraction
I/O	Input and Output
LCS	Least Common Subsumer
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Model
ML	Machine Learning
NL	Natural Language
NLP	Natural Language Processing
PoS	Part-of-Speech
RE	Requirements Engineering
RFP	Request for Proposal
RT	Requirements Traceability
SAO	Subject, Action, Object
SBD	Sentence Boundary Disambiguation
WSD	Word-Sense Disambiguation
WuP	Wu&Palmer algorithm

B.2 Glossary of Part-of-Speech Tags

Acronym	**Definition**
CC	coordinating conjunction
CD	cardinal number
DT	determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective
JJR	adjective, comparative
JJS	adjective, superlative
LS	list marker
MD	modal
NN	noun, singular or mass
NNS	noun plural
NNP	proper noun, singular
NNPS	proper noun, plural
PDT	predeterminer
POS	possessive ending
PRP	personal pronoun
PRP$	possessive pronoun
RB	adverb
RBR	adverb, comparative
RBS	adverb, superlative
RP	particle
TO	to
UH	interjection
VB	verb, base form
VBD	verb, past tense
VBG	verb, gerund/present participle
VBN	verb, past participle
VBP	verb, sing. present, non-3d
VBZ	verb, 3rd person sing. present
WDT	wh-determiner
WP	wh-pronoun
WP$	possessive wh-pronoun
WRB	wh-abverb

B.3 Glossary of Dependencies from the Dependency Parser

Acronym	**Definition**
abbrev	abbreviation modifier
acomp	adjectival complement
advcl	adverbial clause modifier
advmod	adverbial modifier
agent	agent
amod	adjectival modifier
appos	appositional modifier
attr	attributive
aux	auxiliary
auxpass	passive auxiliary
cc	coordination
ccomp	clausal complement
complm	complementizer
conj	conjunct
cop	copula
csubj	clausal subject
csubjpass	clausal passive subject
det	determiner
dobj	direct object
expl	expletive
infmod	infinitival modifier
iobj	indirect object
mark	marker
measure	measure-phrase modifier
neg	negation modifier
nn	noun compound modifier
nsubj	nominal subject
nsubjpass	passive nominal subject
num	numeric modifier
number	element of compound number
parataxis	parataxis
partmod	participial modifier
pcomp	prepositional complement
pobj	object of a preposition
poss	possession modifier
possessive	possessive modifier
preconj	preconjunct
predet	predeterminer
prep/prepc	prepositional modifier
prt	phrasal verb particle
punct	punctuation
purpcl	purpose clause modifier
quantmod	quantifier phrase modifier
rcmod	relative clause modifier
ref	referent
rel	relative

tmod	temporal modifier
xcomp	open clausal complement
xsubj	controlling subject